Netbeans - das universale Werkzeug für Applets, Desktop- und Mobile Applications in JAVA

Leiter: Dr. Karsten Menzel
Teilnehmer: Jan Bongartz, Markus Wollny, Gregor Piel, Max Leopold, Marcel Höft, Tim Ingensiep, Sascha Oedekoven, Christian Kropp, Alexander Nolting, Artur Schmidt, Vladimir Komarov, Alexander Zitzer, Paul Bernasch, David Hamm, Holger Jansen, Dominic Scharmann

Inhalt

Vorwort

An der FH Niederrhein wurde im Wintersemester 2010/2011 mit 18 Studenten ein Seminar zum Thema Netbeans gehalten. Im Rahmen des Seminars wurden verschiedene Themen durch die Studenten erarbeitet, die sich insbesondere auf grundlegende Programmiertechniken beziehen und durch Netbeans unterstützt werden.

Zu der Entwicklungsumgebung (IDE) Netbeans gibt es bislang nur recht rudimentäre Einführungsliteratur in Deutsch. Um ein empfehlenswertes Buch, wie „NetBeans Platform 6 - Rich-Client-Entwicklung mit Java von Heiko Böck“ lesen und verstehen zu können sind sowohl umfangreiche Java als auch Netbeans Kenntnisse erforderlich. Aber ein Buch, welches auch „HalloWelt.java“ in Netbeans erklärt, fehlt bisher. Dies hat möglicherweise dazu geführt, dass in Deutschland Eclipse als Entwicklungsumgebung deutlich mehr verbreitet ist – denn hierzu gibt es eine Vielzahl von Einführungsbüchern in Deutsch. Jedoch ist Netbeans auch unter vielen Entwicklern, die IDE der Wahl und sollte auch deutschsprachigen Studenten noch häufiger nahe gebracht werden.

Ziel des Seminars war es deshalb auch ein seminarbegleitendes Skript zu erstellen, welches ein wenig die Lücke schließt und Einsteigern den Zugang zu Netbeans erleichtern kann.

Netbeans wurde entwickelt und unterstützt von SUN, die auch Entwickler von Java selbst sind. SUN wurde nun durch Oarcle übernommen. Manches ändert sich dadurch in der Java-Umgebung – aber mit Netbeans geht es aber entsprechend weiter. Hier nur einige Gründe für den Erfolg von Netbeans:

- Intuitiver GUI-Designer für Swing (Selbst ohne Swing-Kentnisse kann der User mittels drag & drop tolle Oberflächen erstellen und direkt nutzen)
- Umfangreiches “Refractoring”
- Einer der Besten “Code-Profiler” (Speicher- , CPU- und Performance-Probleme werden transparent)

- UML(Unified Modeling Language)-Unterstützung – hier gibt es aber seit der Version 6.7 Änderungen
- Ant-Integration
- J2ME (nach dem ersten Durchschauen des Einführungstutorial lassen sich direkt kleinere Anwendungen mit Netbeans programmieren)
- Integriertes Update-Center
- Tomcat-Support (kommt im Bundle mit Apache Tomcat. Nach Erstellung einer Web-Application mit JavaServer Pages (jsp) lässt sich diese direkt starten)

1. Einführung

1.1. Softwareentwicklung

Netbeans unterstützt die einzelnen Schritte der Softwareentwicklung. Die Studenten sollen deshalb auch in dem Seminar einige Bausteine der Entwicklung von Software kennen lernen.

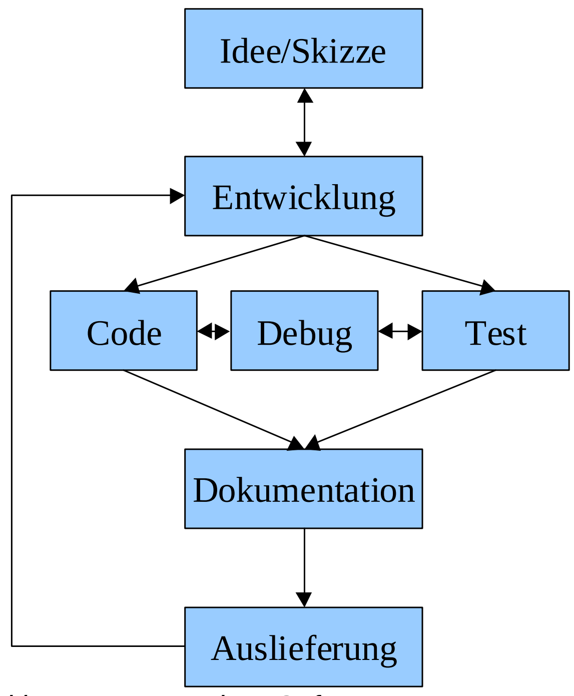


Abb. 1: Typischer Entwicklungsprozess einer Software

1.1.1. Ideenentwicklung mit UML

Bis zur Version 6.7. gab es in Netbeans ein UML-Plugin. Die Entwicklung wurde eingestellt. Stattdessen kann man innerhalb Netbeans nun mit dem Plugin von Visual Paradigma arbeiten.

UML steht für Unified Modelling Language, welche es erlaubt einen graphischen Entwurf von Strukturen und Komponenten im Sinne des objektorientierten Softwaredesigns zu entwerfen. Innerhalb von Netbeans ist dann ein „Forward and Reverse Engineering“ möglich. Alle Diagrammarten werden visuell im Editor erstellt und können in ihren Eigenschaften angepasst werden. Der Entwurfsprozess wird durch vordefinierte Modelle und Entwurfsmuster unterstützt. Typische Beispiele sind:

- Aktivitäten-Diagramm
- Klassen-Diagramm
- Kollaborations-Diagramm
- Komponenten-Diagramm

- Verteilungs-Diagramm
- Interaktions-Diagramm
- Zustands-Diagramm
- Anwendungsfall-Diagramm

1.1.2. Entwicklung von Code

Hier hilft Netbeans durch umfangreiche Editorfunktionen, Codecompletion und Refractoring. Einige Features des Editors sind:
Javatipps, die bei der Quelltexterstellung helfen
Komponentenpaletten zum einfachen Einfügen von Codeschnipseln (z. B. HTML, JSP oder SQL) via Drag & Drop in eine JSP- oder HTML-Seite
Fehlermarkierung, die anzeigt, ob die Datei kompilierbar ist oder nicht, und einen Überblick über alle Fehler, Warnungen, Todos und Bookmarks im Quelltext gibt
Umschließen eines selektierten Statements mit einem try/catch-Block
SQL-Editor mit Syntaxhervorhebung
einfaches Hinzufügen von XML-Dateien, wie Dokumenttypdefinitionen (DTD) und Schemas (XSD), für die Nutzung bei der Code Completion und die Validierung
Einführung einer Möglichkeit, aus vorgegebenen Sets für Tastenkürzel oder Farbschemas (z. B. im Emacs-Stil) zu wählen oder diese nach eigenen Wünschen anzupassen

Codecompletion umfasst beispielhaft folgendes:

- Code von konfigurierbaren Vorlagen
- Getter und Setter
- Anonyme Innere Klassen
- Methodenkörper für Methoden, welche überschrieben oder implementiert werden
- try/catch-Blöcke mit passenden Exceptions
- Zusätzlich kann durch die Verwendung bestimmter Abkürzungen (Camel Case Identifier) Code erzeugt werden. So kann beispielsweise die Eingabe von „AIOOBE“ automatisch durch ArrayIndexOutOfBoundsException ersetzt werden oder „sout“ führt zu System.out.printeln(““);

Refractoring ist ein Umbenennen/Umwandeln bei dem auf den Kontext Rücksicht genommen wird. Beispielhaft sind zu nennen:

- Rename – Umbenennen von Paketen, Klassen, Methoden und Variablen
- Extract Method – Quelltextabschnitte innerhalb von Methoden zu neuen Methoden extrahieren
- Change Method parameters – Signatur einer Methode interaktiv verändern (Parameterreihenfolge ändern)
- Encapsulate Fields – Direktzugriffe auf Objektfelder in Getter- und Setter-Methoden wandeln
- Pull Up – Eine Methode in die Superklasse verschieben
- Push Down – Eine Methode in eine Subklasse verschieben
- Move Class – Eine Klasse in ein anderes Paket verschieben
- Move Inner to Outer Level – Eine innere Klasse in eine normale Klasse wandeln
- Convert Anonymous Class to Inner – Eine anonyme Klasse in eine benannte innere Klasse wandeln
- Extract Interface – Aus der Klassensignatur ein Schnittstellengerüst erzeugen
- Extract Superclass – Aus einer Klasse eine Superklasse extrahieren
- Use Supertype where Possible – Casts auf Superklassen umstellen
- Safe Delete – Testet vor dem Löschen von Feldern, Methoden und Klassen, ob Verwendungen existieren

1.1.3. Fehlersuche

Im nächsten Schritt der Softwareentwicklung und in dem Seminar geht es dann um die Fehlersuche. 2 Vorträge beschäftigen sich mit dem sehr wichtigen Thema Logging. Der Entwickler sollte das Logging zu einem festen Bestandteil der Entwicklung machen. Neben dem Logging in Netbeans wird auch ein Blick auf Log4j geworfen. Anschließend werden Debugger und Profiler behandelt. Der Profiler wird bei den Experten als ein wesentlicher Pluspunkt der IDE Netbeans angesehen.
Er liefert Informationen über das Laufzeitverhalten der Anwendung (mit wenig Aufwand wird Anwendung gemessen).
Folgende Funktionen beherrscht der Profiler:

- Überwachung der CPU-Last, CPU-Geschwindigkeit und CPU-Ausführung
- Speicherbenutzung
- Speicherüberwachung mit Speicherlecktests
- Überwachung von nebenläufigen Prozessen (Threads) und deren Status
- Unterstützung für das JDK 1.4.2, JDK 5.0 und JDK 6.0
- Profiling von laufenden Anwendungen
- Speichern von Profiler-Abbildern für die nachträgliche Bearbeitung
- Profiling von entfernten VMs (über Netzwerkprotokolle)
- Durch Integration in DIE lassen sich sehr einfach Performance- und Speicherprobleme aufspüren.

1.1.4. Tests

Ein wichtiger Schritt parallel zum Entwickeln der Software ist das Testen und viele Autoren meinen sogar, dass der Test vor der Entwicklung stehen sollte. Deshalb werden auch 2 Vorträge des Seminars dem Testen gewidmet.

1.1.5. Versionieren und Dokumentieren

Ohne Versionsverwaltung heute keine Softwareentwicklung mehr. Deshalb ein Blick in dieses umfangreiche Thema:
Durch NetBeans werden folgende Versionskontrollsysteme direkt unterstützt:

- CVS
- SubVersion (SVN)
- Mercurial
- ClearCase (über den Plugin Manager)

Unterstützung fügt sich in den typischen Programmierworkflow ein und versteckt die abstrakte Bedienung mittels bekannter Befehle.
Die IDE erkennt automatisch Arbeitsverzeichnisse und listet alle geänderten Dateien im Versioning Window auf. Die Versionsunterstützung ist ebenfalls in das Projektsystem und in die Refactoring-Funktionen integriert.
Typische Funktionen des Versionskontrollsystems sind direkt über das Kontextmenü zu erreichen. Die Verwaltung von mehrere Repositories und Branches ist auf einfache Weise möglich. Das Versioning Window zeigt einen Überblick über den Status von Dateien, Ordnern, Paketen und Projekten. Mit dem Kommando Diff können Unterschiede zwischen den einzelnen Versionen angezeigt werden. Zusätzlich zu dem Quelltext können auch Projektmetadaten verteilt werden.

Zusätzlich zu den entfernten Versionkontrollsystemen führt NetBeans für jeden Quelltext eine so genannte Local History. Über diese ist es möglich auch ohne die Unterstützung einer Versionskontrolle Änderungen am Quelltext zu verfolgen und gegebenenfalls rückgängig zu machen. Zum Schluss wird dann noch ein Blick auf „Kenai“ geworfen, die noch existierende Web-Plattform für gemeinsame Javacode-Entwicklung.

Daran schließt sich die Dokumentation mit Javadoc an.

Javadoc ist ein Software-Dokumentationswerkzeug, das aus Java-Quelltexten automatisch HTML-Dokumentationsdateien erstellt. Es wurde ebenso wie Java von Sun Microsystems entwickelt und ist seit Version 2 ein Bestandteil des Java Development Kits.
Die Dokumentation kann somit durch spezielle Kommentare im Quelltext erstellt werden. Dadurch können Beschreibungen für Interfaces, Klassen, Methoden und Felder über spezielle Doclet-Tags definiert werden.
Netbeans generiert automatisch Dokumentationsseiten in HTML.

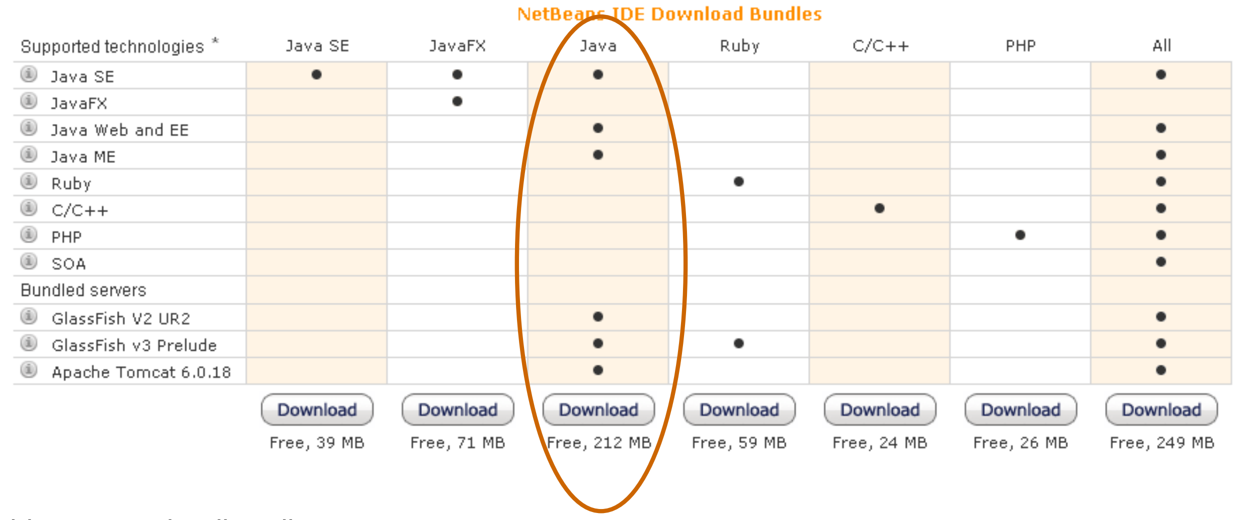
1.1.6. Ist dies alles?

Nein, die Netbeans IDE unterstützt bei vielen weiteren Softwareentwicklungsschritten. Im Seminar kann hier nur ein erster Eindruck gegeben werden und es schließen sich Themen, wie

- GUI-Editor,
- Desktop Applications,
- Datenbankanbindung,
- Internationalisierung,
- Web-Applications,
- Javen,
- und zuletzt Mobile Applications an.

1.2. Installation

Java und Netbeans lassen sich in einem Schritt von der Website www.netbeans.org herunterladen:



NetBeans IDE Download Bundles

Supported technologies *	Java SE	JavaFX	Java	Ruby	C/C++	PHP	All
Java SE	•	•	•				•
JavaFX		•					
Java Web and EE			•				•
Java ME			•				•
Ruby				•			•
C/C++					•		•
PHP						•	•
SOA							•
Bundled servers							
GlassFish V2 UR2			•				•
GlassFish v3 Prelude			•	•			•
Apache Tomcat 6.0.18			•				•
	Download	Download	Download	Download	Download	Download	Download
	Free, 39 MB	Free, 71 MB	Free, 212 MB	Free, 59 MB	Free, 24 MB	Free, 26 MB	Free, 249 MB

Abb. 2: Downloadbundle

Für das Seminar empfiehlt sich das gezeigt Bundle der IDE zu installieren. Zum Zeitpunkt des Seminars war dies Version 6.9.1.
Um Netbeans zu installieren muss zunächst das Java Development Kid installiert werden (automatisch im Bundle).

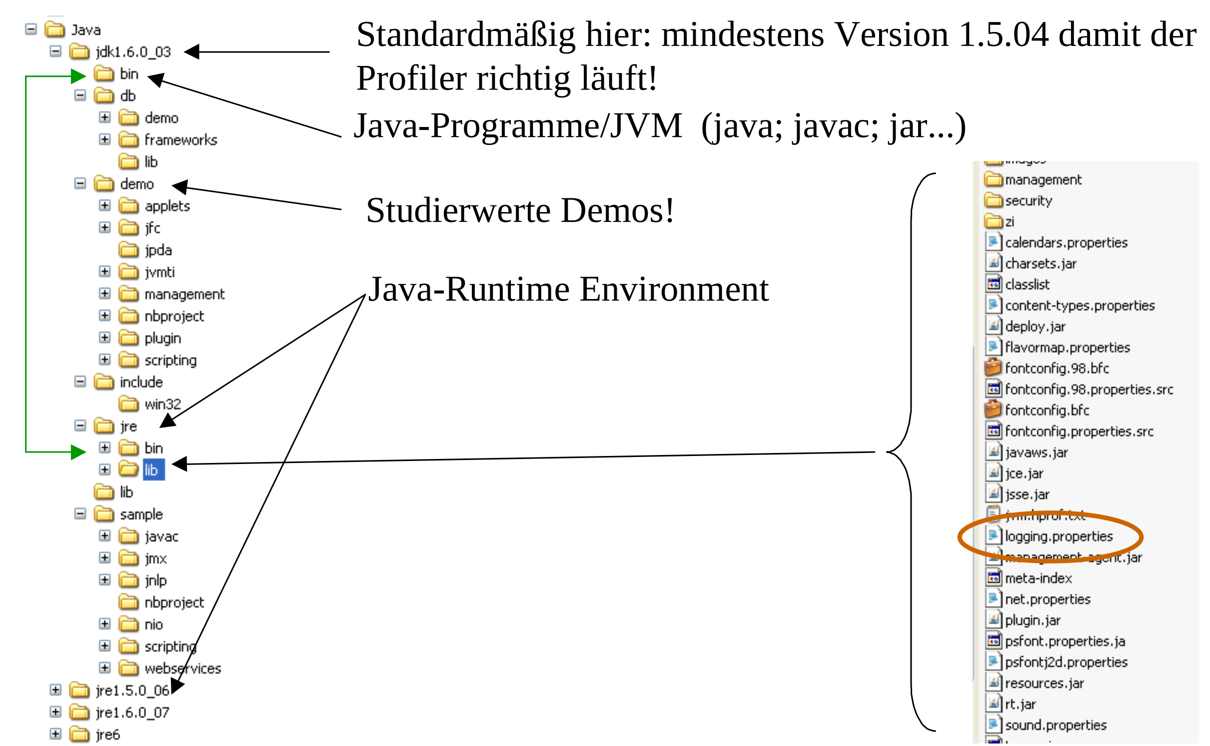


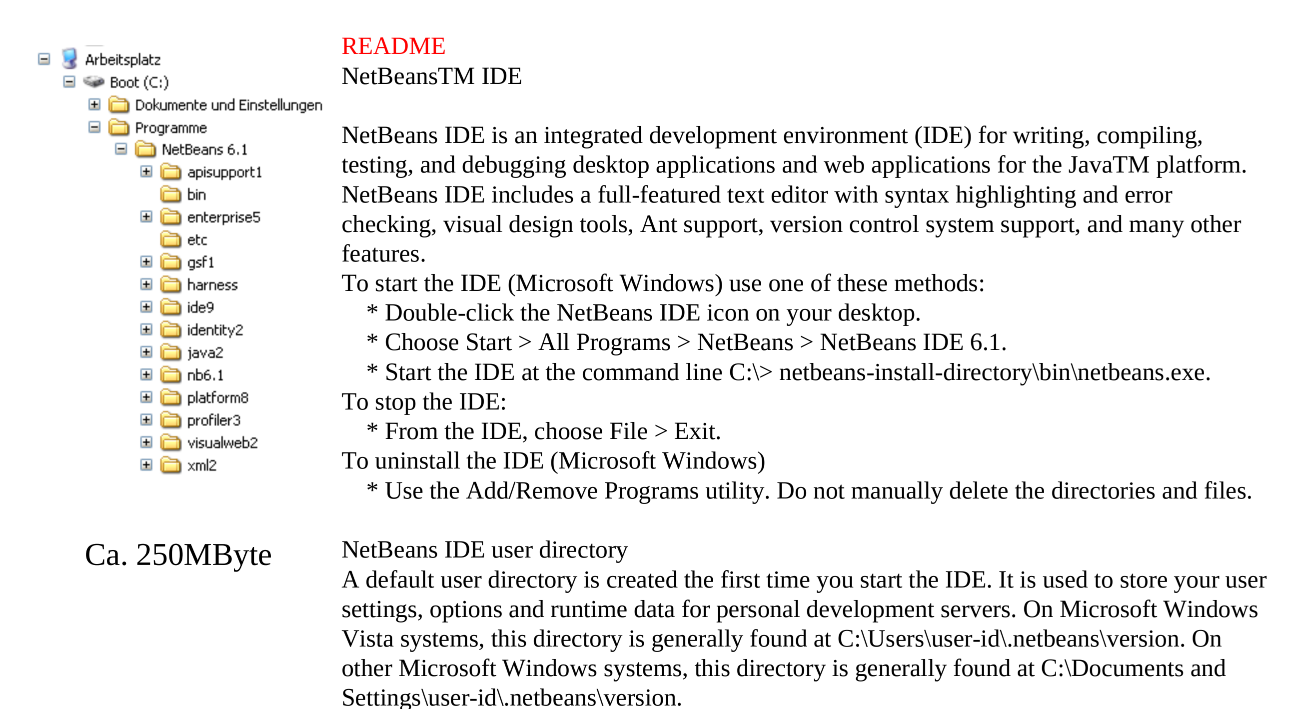
Abb. 3:Unter Windows werden durch die Installation typischerweise die gezeigten Ordner angelegt. Im Seminar werden wir die Datei logging.properties noch brauchen.

Das JDK beinhaltet neben der Laufzeitumgebung (JRE) folgende Java-Entwicklungswerkzeuge:

- javac (Java-Compiler; Java-Quellcode in Java-Bytecode *.class)
- javadoc (zur Erstellung der Dokumentation; aus Quellcode werden html-Seiten)
- jar (zur Erstellung von Java-Archive; enthält sämtliche .class-Dateien, Grafiken, Textdateien und sonstige Dateien)
- jarsigner (zur Signierung von Java-Anwendungen oder Bibliotheken)
- htmlconverter (Java Plug-in HTML Converter)
- appletviewer (ist sehr einfacher Browser, der nur die in einem HTML-Dokument enthaltenen Java-Applets anzeigt, und zwar jedes in einem eigenen Fenster =>zum Testen von Applets bei der Programmierung)

Die aktuelle Version des JDK war zum Seminar 6 Update 21.

Die Installation der Netbeansumgebung erfolgt versionsbezogen (je Version) in separate Ordner.



README
NetBeansTM IDE

NetBeans IDE is an integrated development environment (IDE) for writing, compiling, testing, and debugging desktop applications and web applications for the JavaTM platform. NetBeans IDE includes a full-featured text editor with syntax highlighting and error checking, visual design tools, Ant support, version control system support, and many other features.
To start the IDE (Microsoft Windows) use one of these methods:
* Double-click the NetBeans IDE icon on your desktop.
* Choose Start > All Programs > NetBeans > NetBeans IDE 6.1.
* Start the IDE at the command line C:\> netbeans-install-directory\bin\netbeans.exe.
To stop the IDE:
* From the IDE, choose File > Exit.
To uninstall the IDE (Microsoft Windows)
* Use the Add/Remove Programs utility. Do not manually delete the directories and files.

NetBeans IDE user directory
A default user directory is created the first time you start the IDE. It is used to store your user settings, options and runtime data for personal development servers. On Microsoft Windows Vista systems, this directory is generally found at C:\Users\user-id\.netbeans\version. On other Microsoft Windows systems, this directory is generally found at C:\Documents and Settings\user-id\.netbeans\version.

Abb. 4: Ordnerstruktur der Version 6.1im Programmordner und zugehörige Readme-datei.

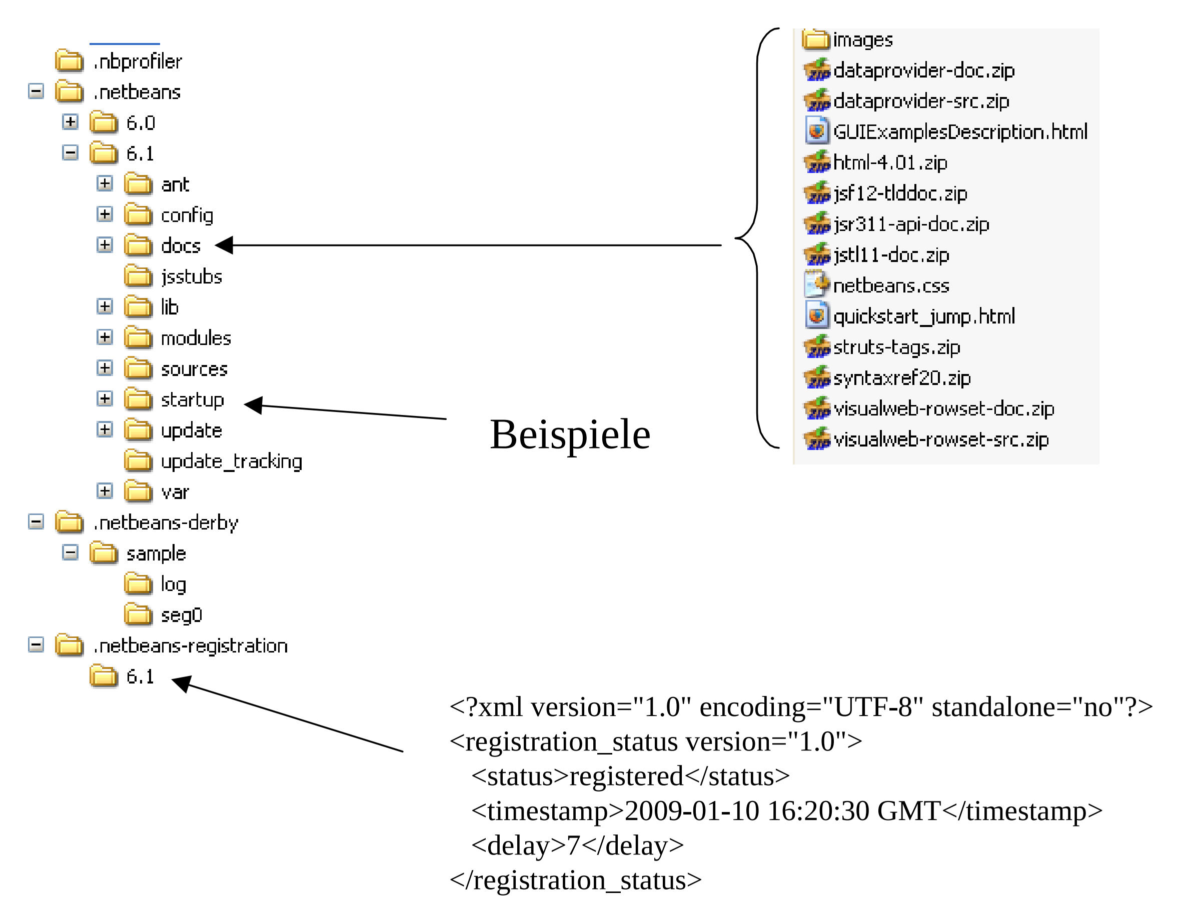


Abb. 5: Ordnerstruktur im Ordner des Users

1.3. Beispiel& Übungsprogramm

Um an einer kleinen Anwendung die im Seminar auftauchenden Programmiertechniken üben zu können, wurde gemeinsam ein kleines Programm entwickelt. Das Programm soll im Wesentlichen zur Synchronisierung von Fotodateien dienen, die mit 2 verschiedenen Fotoapparaten zum gleichen Ereignis erstellt wurden. Wurde vorher die Zeit nicht

synchronisiert, so lassen sich hinterher die Fotos kaum noch in chronologische Reihenfolge bringen. Um den Studenten den Grundcode abzunehmen wurde eine Rohfassung mit vielen Verbesserungsmöglichkeiten zur Verfügung gestellt. Einige Verbesserungen werden insbesondere am Schluss dieses Skripts beschrieben.

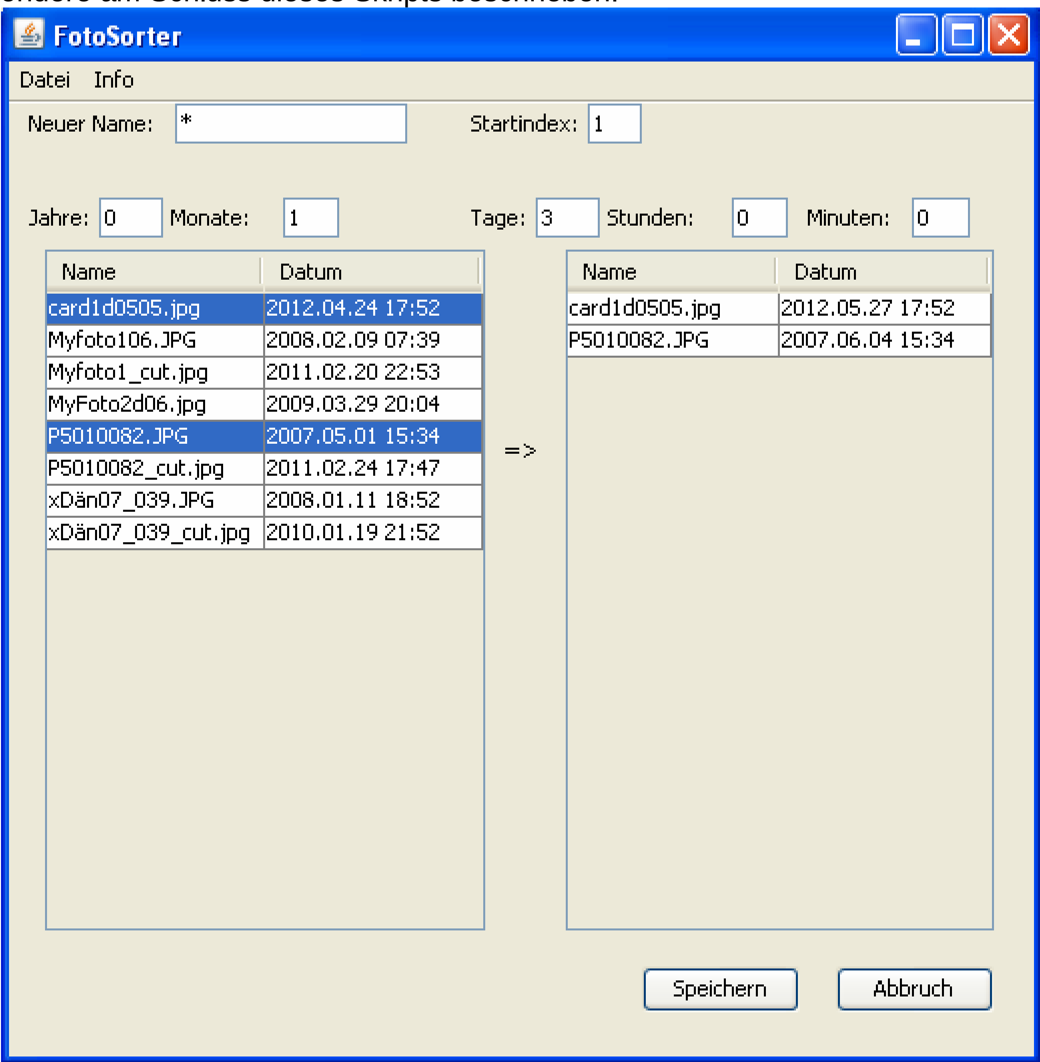
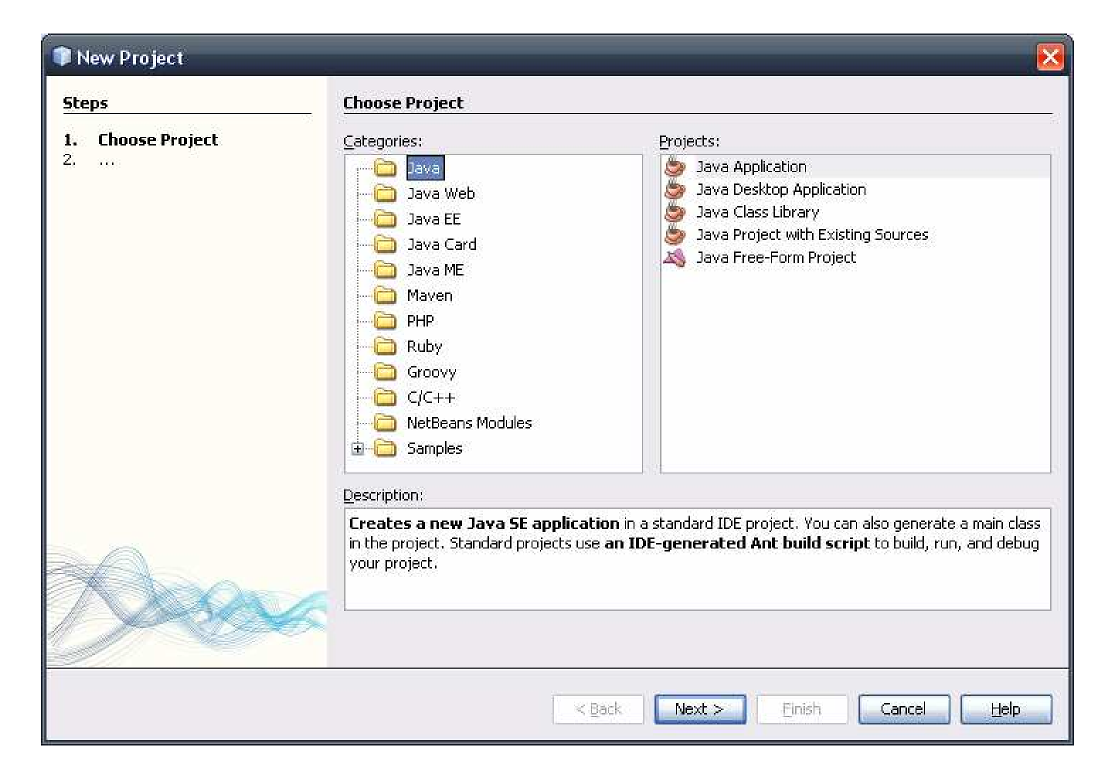


Abb. 6: GUI des Beispielprogramms

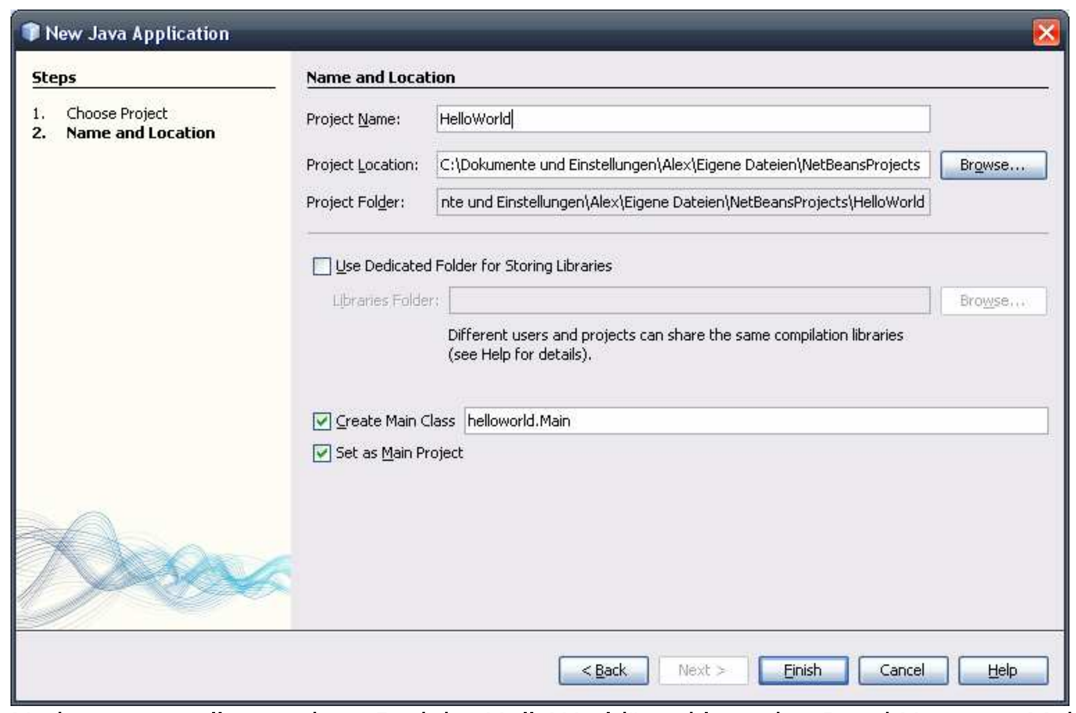
2. Hello World mit Netbeans – Alexander Zitzer

Mit nur wenigen Klicks können wir ein kleines Programm erstellen, welches uns im Konsolenfenster „Hello World“ ausgibt.
Dazu gehen wir wie folgt vor:

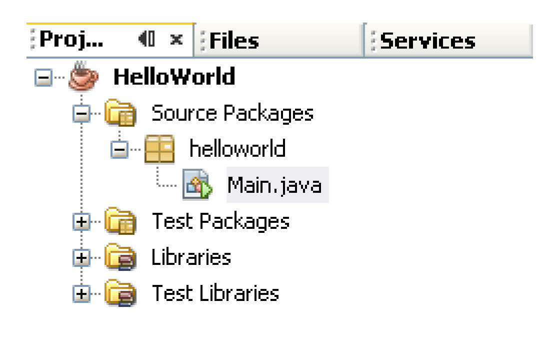
Netbeans starten -> File -> New Project…
Im folgenden Fenster wählen wir unter Categories „Java“ und unter Projects wählen wir „Java Application“ und klicken auf „Next“



Jetzt können wir unserem Projekt einen Namen vergeben



Netbeans erstellt uns einen Projekt „HelloWorld“ und legt eine Datei Namens „Main.java“ an.



In der Klasse „Main.java“ ist nun schon ein Coderumpf inclusive einer Methode „Main“ angelegt, wo wir unseren Code eintragen können.

```
 6  package helloworld;
 7
 8  /**
 9   *
10   * @author Alex
11   */
12  public class Main {
13
14      /**
15       * @param args the command line arguments
16       */
17      public static void main(String[] args) {
18          // TODO code application logic here
19      }
20
21  }
22
```

Hier geben wir nun „`System.out.println("Hallo Welt");`“ ein und unser Programm ist fertig.

```
17      public static void main(String[] args) {
18          System.out.println("Hallo Welt");
19      }
```

Nachdem wir mit F6, oder mit dem Button „Run Main Project“ das Programm starten, können wir unsere Ausgabe schon sehen:

```
Output - HelloWorld (run)
run:
Hello World
BUILD SUCCESSFUL (total time: 0 seconds)
```

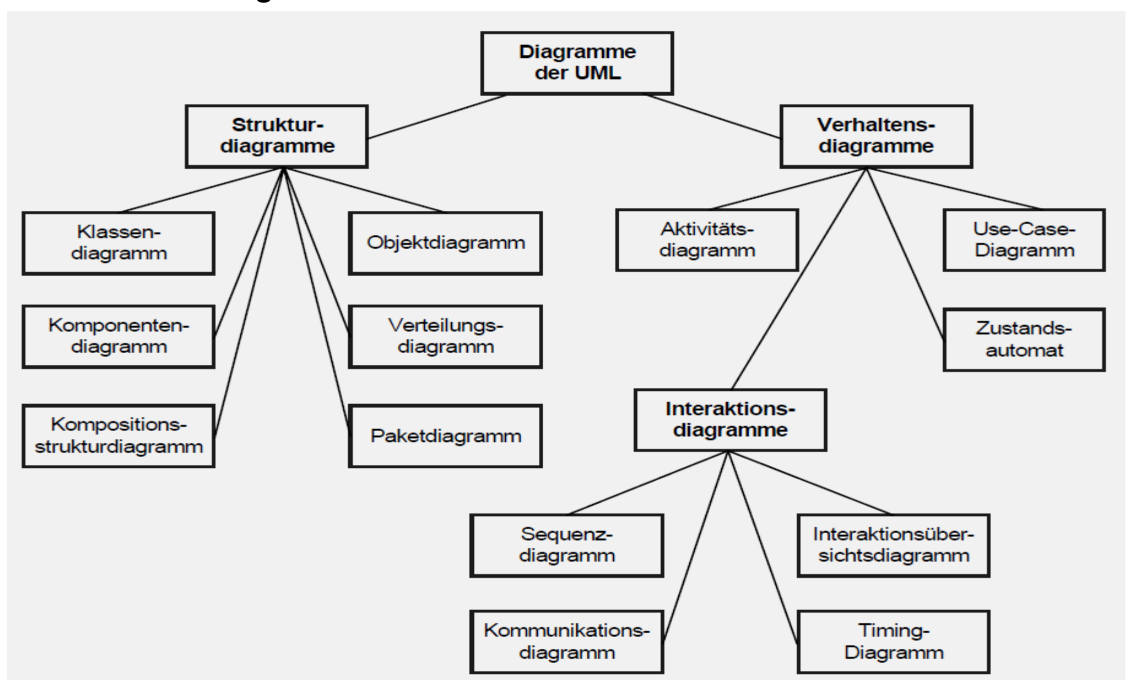
Und hiermit ist unser kleines Programm fertig.

3. UML - Einführung & Beispiele - Jan Bongartz, 25.10.2010

3.1. Was ist UML?

Die Abkürzung UML steht für Unified Modeling Language, was auf Deutsch so viel hieß wie „Vereinheitlichte Modellierungssprache“.
Diese Sprache definiert wichtige Bezeichner und Beziehungen die der Modellierung und Beschreibung von Software-Abläufen und -Systemen dienen. Diese zum Teil auch grafische Abbildungen finden, im Gegensatz zu Programmiersprachen, auf einer hohen Abstraktionsebene statt.
Die dabei definierten Begriffe lassen sich aufteilen in einerseits grafische Notationen für Modelle und andererseits in Beziehungen von Objekten untereinander.
Zu den grafischen Notationen zählen unter Anderem Klassendiagramme zur Darstellung von statischen Strukturen oder Aktivitätsdiagramme zur Veranschaulichung von dynamischen Abläufen.
Zu den Beziehungen von Objekten werden Begriffe für Vererbung, Realisierung und Abhängigkeiten festgelegt.

3.2. Entstehung



Die in den 1990er Jahren immer zunehmende Bedeutung und breitere Verwendung von Objektorientierter Softwareentwicklung machte eine einheitliche Modellierungssprache immer wichtiger. Die drei Informatiker Grady Booch, Ivar Jacobson und James Rumbaugh, auch die drei „Amigos“ genannt, entwickelten daher bereits eigene Modellierungssprachen die sie dann strukturiert zusammenführten. So entstand ein Vorläufer der UML 1.
Die Weiterentwicklung von UML wurde an die Object Management Group (OMG), ein 1989 gegründetes Konsortium, das sich mit der Entwicklung von Standards für die herstellerunabhängige systemübergreifende Objektorientierte Programmierung beschäftig, übergeben. Dort wurde am 19. November 1997 UML 1 als Standard akzeptiert. Im Jahr 2005 erschien die überarbeitete Version UML 2 und seit Mai 2010 gibt es Version 2.3.

3.3. UML-Diagramme

Von den 13 verschiedenen UML-Diagrammen werde Ich nachfolgend das Klassendiagramm als ein Beispiel für ein Strukturdiagramm und das Aktivitätsdiagramm sowie das Sequenzdiagramm als Beispiele für Verhaltensdiagramme genauer vorstellen.

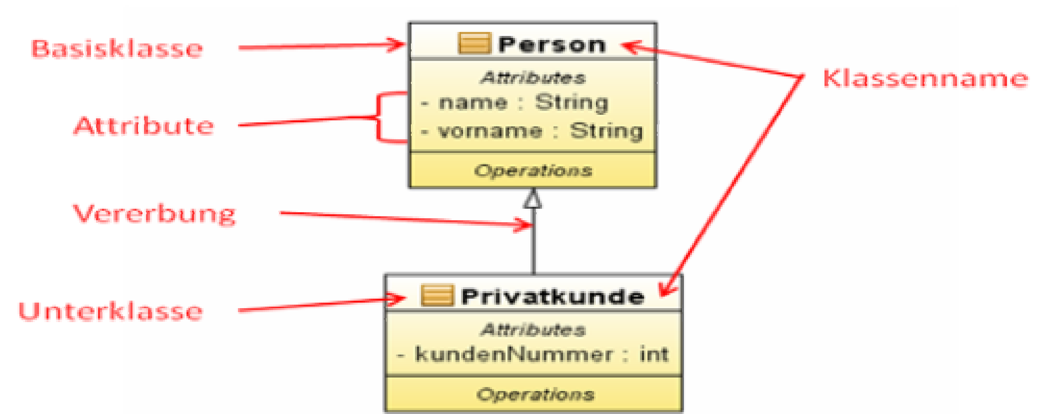
3.3.1. Klassendiagramm

Das Klassendiagramm hilft bei der Modellierung von Klassen, Schnittstelle sowie deren Beziehungen.
Dabei ist die Bezeichnung Klasse ein abstrakter Oberbegriff für die Struktur und das Verhalten eines Objekts. Eine Klasse wird in einem Klassendiagramm durch ein Rechteck dargestellt welches mindestens den Klassennamen aber auch Namen und Typ bzw. Parameter von Attributen und Methoden enthält.
Die Attribute und Methoden einer Klasse können dabei eine unterschiedliche Sichtbarkeit besitzen. So bezeichnet das Schlüsselwort *public*, welches durch das „+“ Zeichen vor dem Attribut oder der Methode im Klassendiagramm dargestellt wird, einen unbeschränkten Zugriff auch von außerhalb der Klasse auf dieses Element. Bei den Schlüsselwörtern *protected* und *private*, dargestellt durch ein „#“ bzw. „-„ Zeichen, kann der Zugriff nur aus der Klasse selbst, bzw. bei protected auch aus einer Unterklasse, erfolgen. Die Sichtbarkeit innerhalb eines Pakets ist durch das Schlüsselwort *package*, dargestellt durch ein „~“ Zeichen, geregelt.

Beziehungen zwischen Klassen

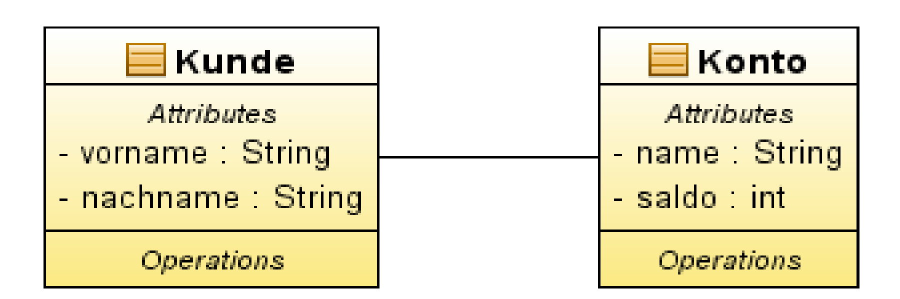
Vererbung

Bei Klassen spricht man von einer Vererbung, wenn eine Klasse, genannt Unterklasse, alle Eigenschaften und Attribute einer anderen Klasse, genannt Ober- oder Basisklasse, übernimmt.
Die Unterklasse kann darüber hinaus die Funktionen der Basisklasse durch weitere Attribute oder Methoden erweitern.

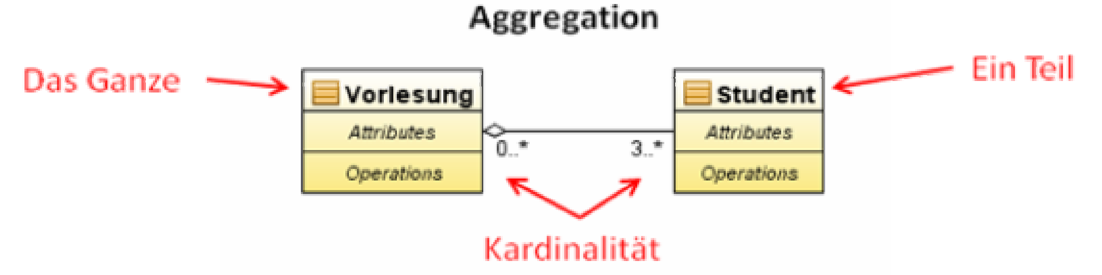


Eine Vererbung wird im UML Kassendiagramm durch einen Pfeil mit einer leeren Spitze von der Unterklasse zur Basisklasse dargestellt.

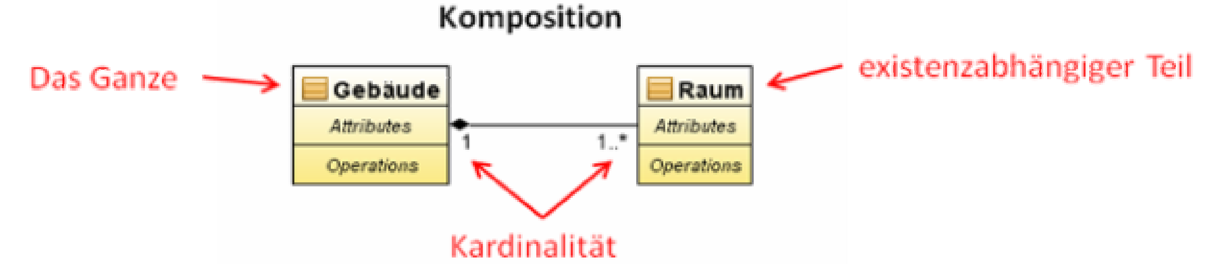
Assoziation

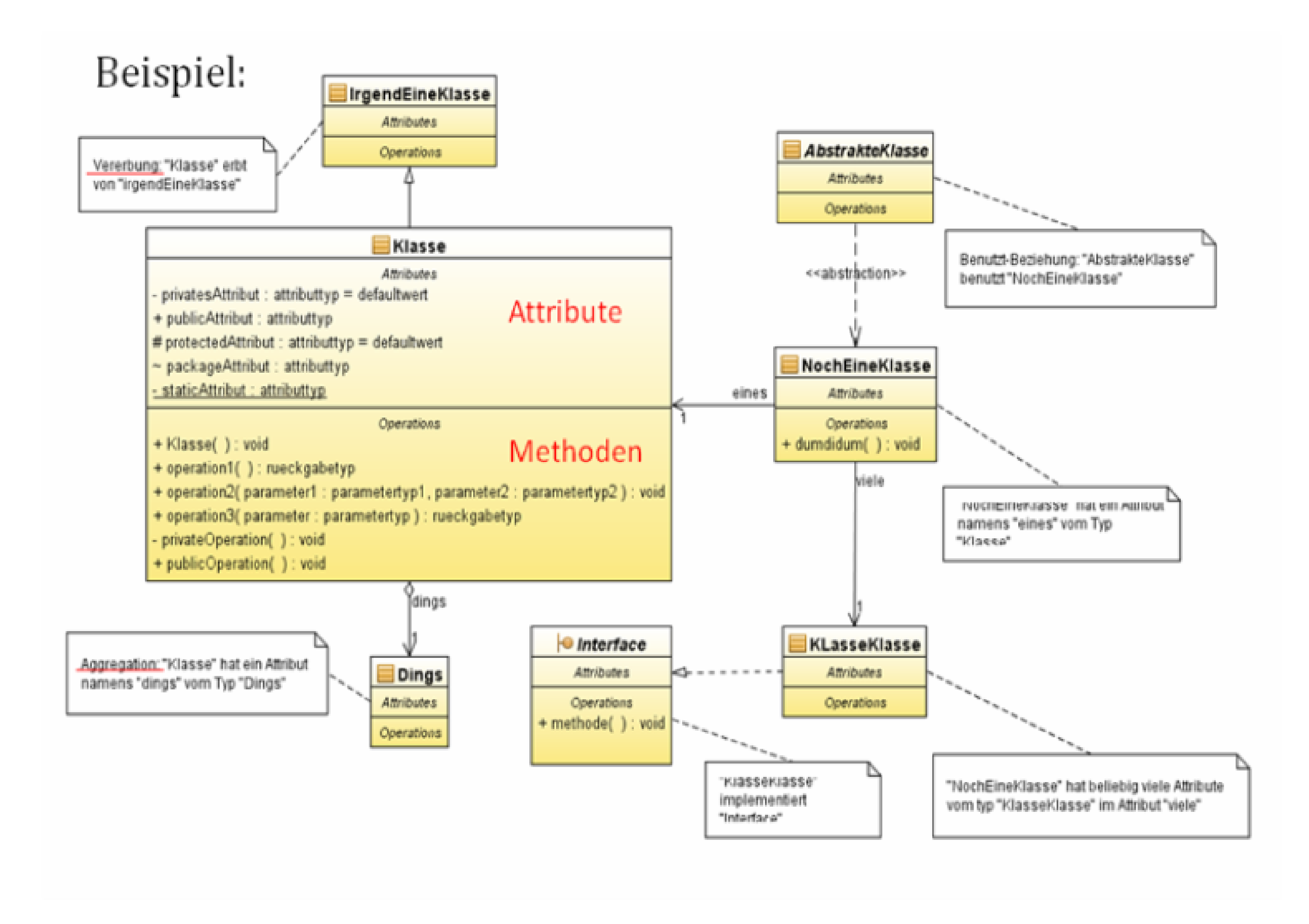


Bei der Assoziation handelt es sich um eine Art „Kennt-Ein-Beziehung“ bei der das Verhältnis zwischen zwei oder mehreren Klassen verdeutlicht wird. Im Klassendiagramm wird die Assoziation durch einen Strich zwischen den Klassen dargestellt. Die Anzahl wird dabei durch Kardinalitäten an beiden Enden des Striches vermerkt.



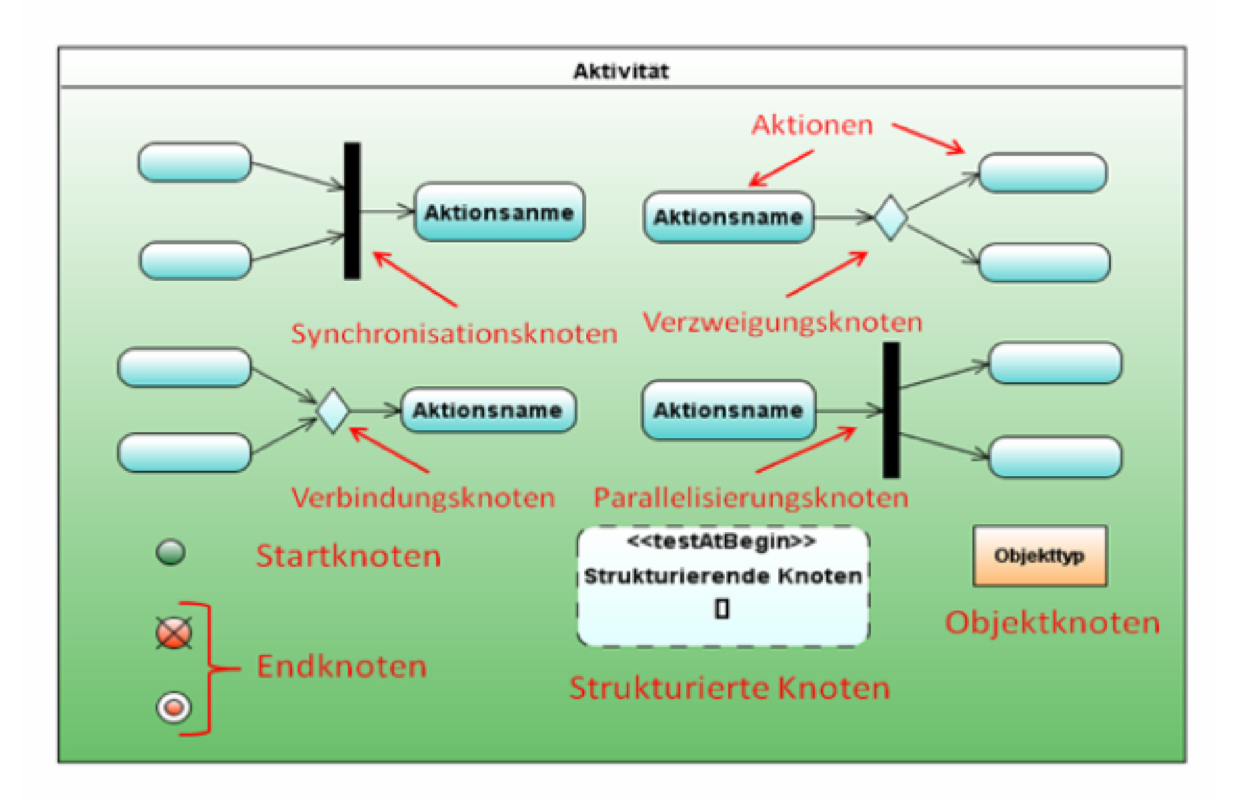
Zwei Spezialfälle der Assoziation sind die Aggregation und die Komposition. Bei beiden handelt es sich um eine „Teile-Ganzes-Beziehung“ jedoch sind bei der strengeren Komposition die Teile vom Ganzen existenzabhängig. Aggregation und Komposition werden im Klassendiagramm durch einen Strich zwischen den Klassen mit einer leeren Raute am Ende des Ganzen dargestellt, wobei bei der Komposition die Raute gefüllt ist.



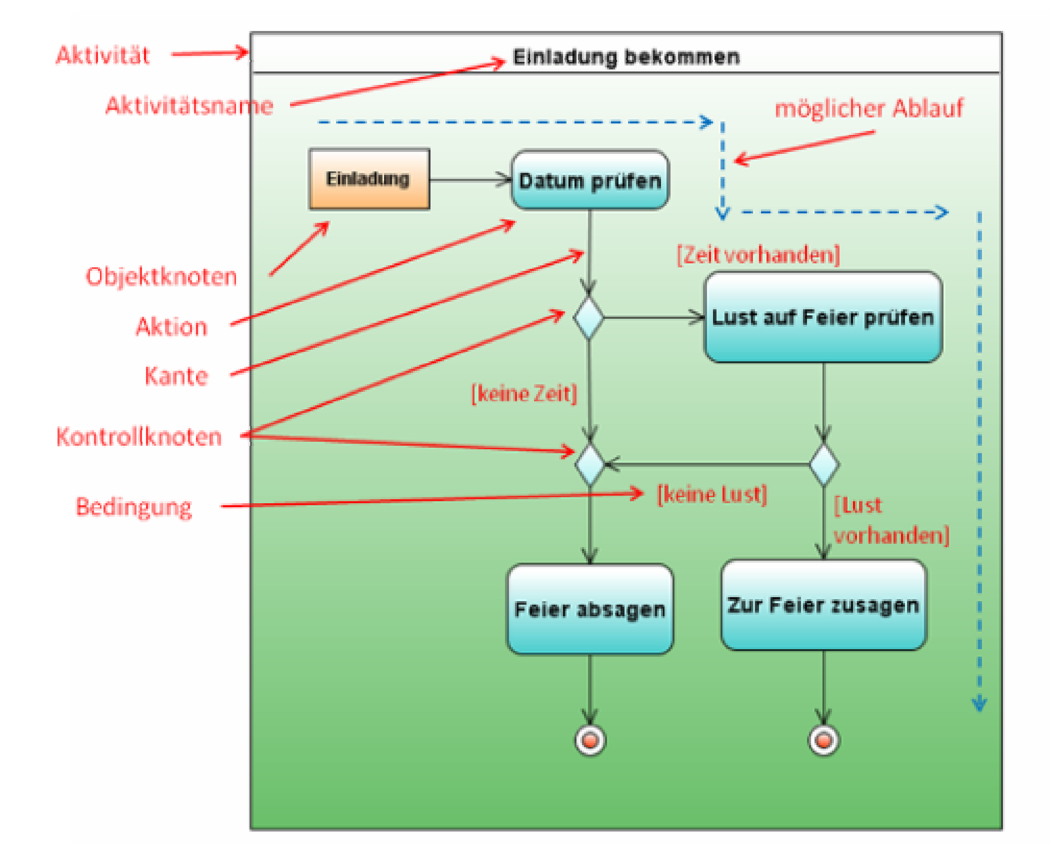


3.3.2. Aktivitätsdiagramm

Ein Aktivitätsdiagramm modelliert das dynamische Verhalten eines Systems anhand einer oder mehrerer Aktivitäten die auch ineinander geschachtelt sein können. Eine Aktivität setzt sich dabei aus einzelnen Aktionen zusammen, welche in Form eines Graphen angeordnet sind. Die Knoten dieses Graphen stehen dabei für die einzelnen Aktionen und Kanten für die Objekt- und Kontrollflüsse.


Aktivität
Aktionen
Aktionsanme
Aktionsname
Synchronisationsknoten
Verzweigungsknoten
Aktionsname
Aktionsname
Verbindungsknoten
Parallelisierungsknoten
Startknoten
<<testAtBegin>>
Strukturierende Knoten
Objekttyp
Objektknoten
Endknoten
Strukturierte Knoten

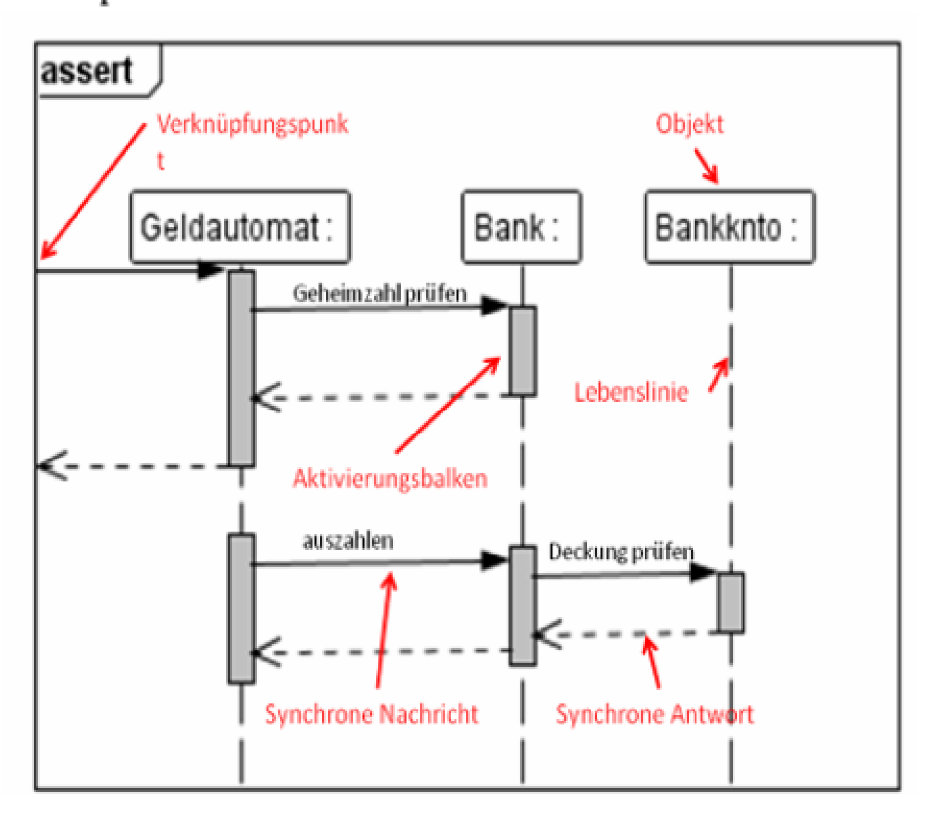
Beispiel:


Aktivität
Einladung bekommen
Aktivitätsname
möglicher Ablauf
Einladung
Datum prüfen
Objektknoten
[Zeit vorhanden]
Aktion
Lust auf Feier prüfen
Kante
[keine Zeit]
Kontrollknoten
Bedingung
[keine Lust]
[Lust vorhanden]
Feier absagen
Zur Feier zusagen

3.3.3. Sequenzdiagramm

Ein Sequenzdiagramm stellt den Informationsfluss zwischen Kommunikationspartnern grafisch dar. Die vertikale Zeitachse symbolisiert dabei den temporalen Ablauf auf der die Interaktionen und Nachrichten entsprechend ihrer zeitlichen Reihenfolge eintreten. Nachrichten werden dabei durch Pfeile dargestellt, dessen Pfeilspitzen bei synchronen Nachrichten gefüllt und bei asynchronen leer sind. Der Aktivierungsbalken unterhalb eines Objektes gibt an, wann ein Objekt über den Kontrollfluss verfügt.

Beispiel:



3.4. Quellen

- http://www.highscore.de/uml/
- http://www.jeckle.de/
- http://www.oio.de/public/objektorientierung/uml-referenz-2-1/index.htm
- http://www.oose.de/swe/uml.html
- http://www3.informatik.uni-erlangen.de/
- http://www.st.informatik.tu-darmstadt.de/
- http://de.wikipedia.org/wiki/Unified_Modeling_Language
- http://www.uml.org/

4. UML mit Netbeans – Markus Wollny, 25.10.2010

4.1. Vorwort

Dieses Skript dient dem Einstieg in die Benutzung von UML mit Netbeans. Es wird vorausgesetzt die einzelnen Diagrammarten von UML verstanden zu haben und deren einzelne Funktionen, wie z.B. die verschiedenen Beziehungen von Elementen zueinander, verwenden zu können.

4.2. Ideenentwicklung mit UML

Bis zur Version 6.7. gab es in Netbeans ein UML-Plugin. Die Entwicklung wurde eingestellt. Stattdessen kann man innerhalb Netbeans nun mit dem Plugin von Visual Paradigma arbeiten.

UML steht für Unified Modelling Language, welche es erlaubt einen graphischen Entwurf von Strukturen und Komponenten im Sinne des objektorientierten Softwaredesigns zu entwerfen. Innerhalb von Netbeans ist dann ein „Forward and Reverse Engineering“ möglich. Alle Diagrammarten werden visuell im Editor erstellt und können in ihren Eigenschaften angepasst werden. Der Entwurfsprozess wird durch vordefinierte Modelle und Entwurfsmuster unterstützt.
Typische Beispiele sind:

- Aktivitäten-Diagramm
- Klassen-Diagramm
- Kollaborations-Diagramm
- Komponenten-Diagramm
- Verteilungs-Diagramm
- Interaktions-Diagramm
- Zustands-Diagramm
- Anwendungsfall-Diagramm

4.3. Installation

4.3.1. Installation von UML bis Netbeans 6.7

Bis zur Netbeans Version 6.7 gestaltet sich die Installation von UML relativ einfach.
Hier ein Beispiel mit der Netbeans IDE 6.5:

Wählen Sie unter dem Menüpunkt “Tools“ die Option “Plugins“ aus.
Nun ist folgendes Fenster zu sehen:

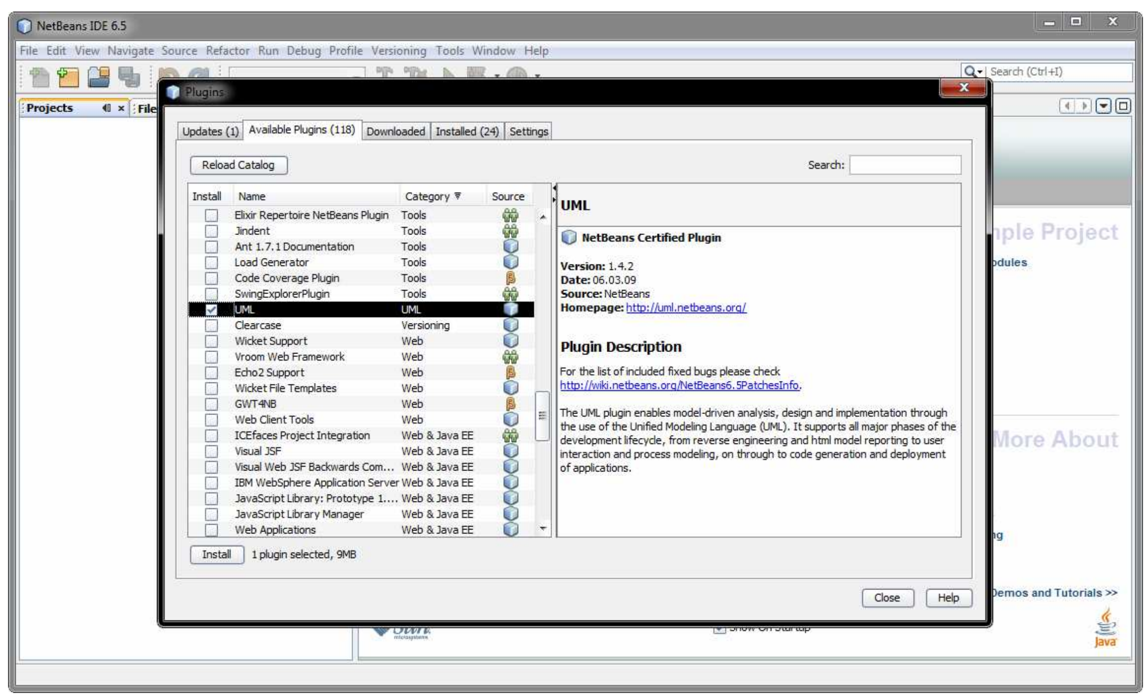


Abb. 7: Plugins-Fenster von Netbeans IDE 6.5

Hier suchen Sie unter dem Reiter “Available Plugins“ UML und wählen diesen an. Danach auf den “Install“-Button klicken.

4.3.2. Installation von UML ab Netbeans 6.8

Ab Netbeans IDE 6.8 wird das UML Plugin leider nicht mehr unterstützt.
Als Ersatz kann man ein Plugin von Visual Paradigm nutzen,
welches Sie hier finden:

http://www.visual-paradigm.com/product/sde/nb/

Hier laden Sie sich die Community Version runter, welche Sie nach einer Registrierung mit wenigen Einschränkungen für den privaten Gebrauch nutzen können. In diesem Skript wird lediglich der Gebrauch des UML-Plugins bis zur Netbeans IDE 6.7 erklärt, da der Umfang und die gebotenen Funktionen des Plugins von Visual Paradigm die Ausmaße des Skriptes sprengen würde.

4.4. Erstellung eines Projekts mit Arbeitsbereichsübersicht

Im folgenden Kapitel erfahren Sie, wie Sie ein neues UML Projekt erstellen und den Diagrammtypen auswählen. Weiterhin wird der Arbeitsbereich kurz veranschaulicht.

4.4.1. Erstellen eines Projektes mit UML

Nachdem Sie Netbeans gestartet haben, erstellen Sie ein neues Projekt. Dazu gehen Sie unter dem Menüpunkt “File“ auf “New Project“. Nun ist folgendes Fenster zu sehen:

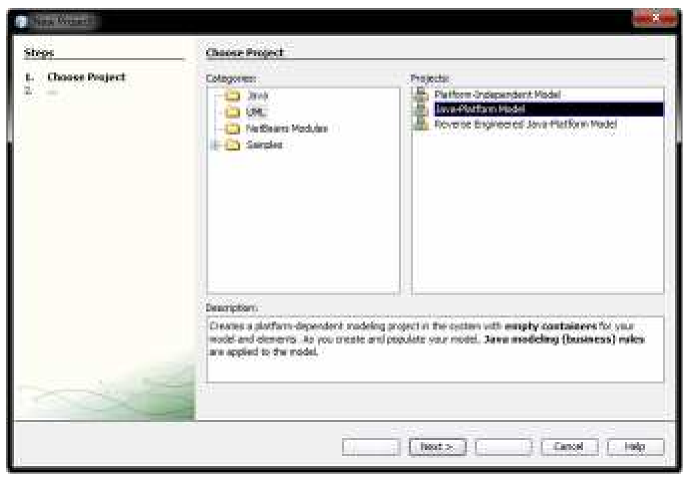


Abb. 8: New Project-Fenster

Wählen Sie nun die Option “UML“ unter “Categories“ und “Java-Platform Model“ unter “Projects“. Bestätigen Sie mit dem “Next“-Button, worauf folgendes Fenster erscheint:

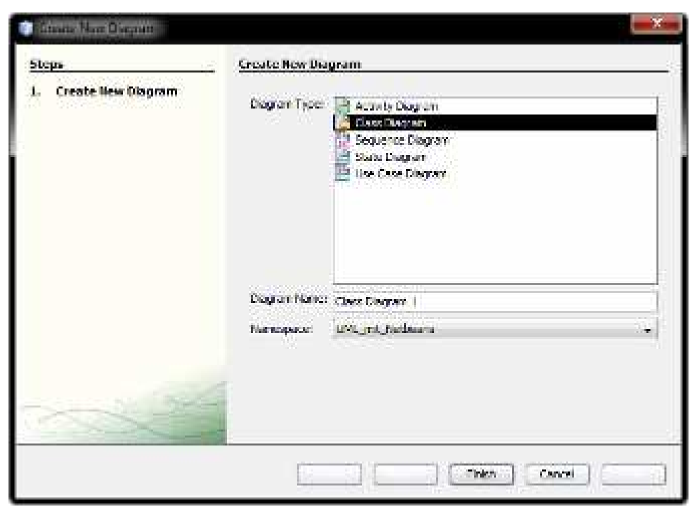


Abb. 9: Create New Diagram-Fenster

Hier können Sie nun zwischen verschiedenen Diagrammarten wählen. Nachdem Sie sich für eine Diagrammart entschieden haben, müssen Sie nur noch mit dem “Finish“- Button bestätigen.

4.4.2. Der Arbeitsbereich

Lassen Sie uns einen kurzen Blick auf Ihren Arbeitsbereich werfen, nachdem Sie Kapitel 3.1 beendet haben:

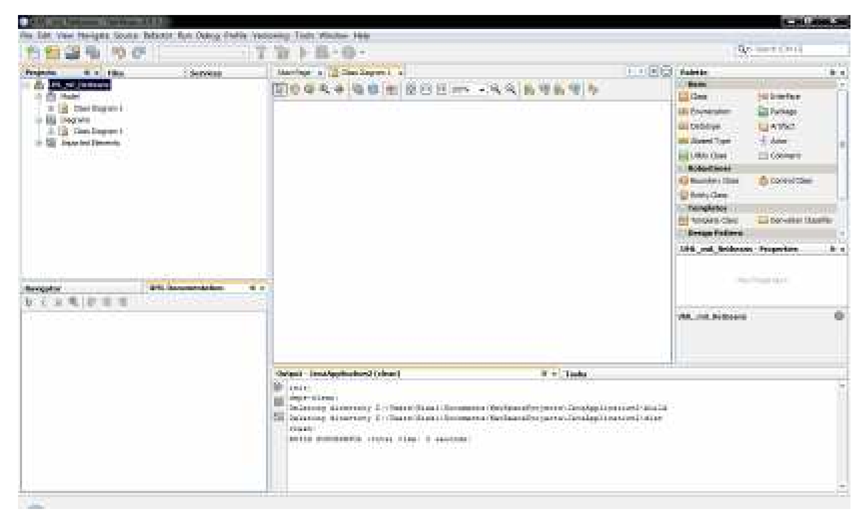
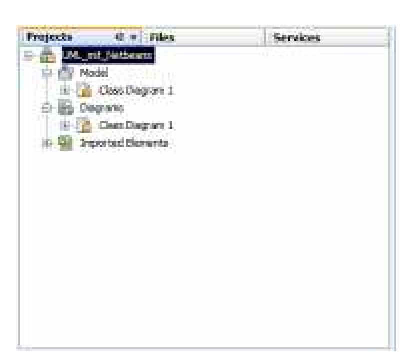


Abb. 10: Der Arbeitsbereich

Nun die einzelnen Bereiche im Detail:

4.4.3. Das Projects-Fenster



Unter "Model" finden Sie ihr Diagramm mit einer vollständigen Auflistung und nähere Informationen zu den einzelnen Elementen.

Unter "Diagrams" sind nur die Diagramme aufgeführt, somit besteht die Möglichkeit schneller auf die einzelnen Diagramme zugreifen zu können (falls Sie mehrere Diagramme in einem Projekt haben, erspart Ihnen dies vielfaches klicken durch den "Model"-Pfad).

4.4.4. Das Properties-Fenster

In diesem Fenster werden Ihnen die Eigenschaften detailliert angezeigt.

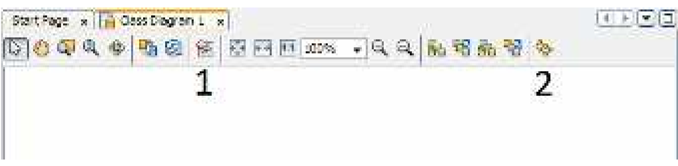


4.4.5. Das Palette-Fenster

Im Palette-Fenster werden Ihnen, je nach gewählter Diagrammart, die verfügbaren Elemente angezeigt, die Sie für Ihr Diagramm benutzen können.
Des Weiteren haben Sie auch die Möglichkeit die Palette durch Schablonen zu erweitern, um so immer wiederkehrende Probleme nicht jedes Mal neu erstellen zu müssen.

4.4.6. Das Arbeitsfenster

Dies ist Ihr Arbeitsbereich, in den Sie die einzelnen Elemente der verschiedenen Diagrammarten hineinziehen:



Neben den üblichen Funktionen wie vergrößern, verkleinern, verschieben u.s.w., gibt es hier 2 weitere, sehr nützliche Funktionen:

1. Export Image: Mit dieser Funktion erhalten Sie die Möglichkeit ihr Diagramm als Bilddatei zu exportieren.

2. Hierarchical Layout: Diese Funktion ordnet Ihr Diagramm nach der hierarchischen Struktur. Vorsicht!!! Die Veränderungen, die die Funktion an Ihrem Diagramm vornimmt, können nicht rückgängig gemacht werden.

4.5. Beispiele

Im folgendem Kapitel finden Sie Beispiele zum Erstellen verschiedener Diagrammtypen unter Netbeans IDE 6.5.

4.5.1. Erstellen eines Klassen-Diagramms

Zunächst wird ein neues Projekt erstellt und als Diagrammtyp “Class Diagram“ gewählt. Danach werden 3 Class-Elemente aus der Palette in das Arbeitsfenster gezogen.



Abb. 11: Arbeitsfenster mit 3 Class-Elementen

Durch Doppelklick auf “Unnamed“ können Sie den Klassen Namen geben.
Attribute und Funktionen können durch einen Linksklick auf das entsprechende Element und der Wahl von “Create Attribute“ und “Create Operation“ vergeben werden. Im Folgenden wurden den Kindern und Eltern Attribute zugewiesen:

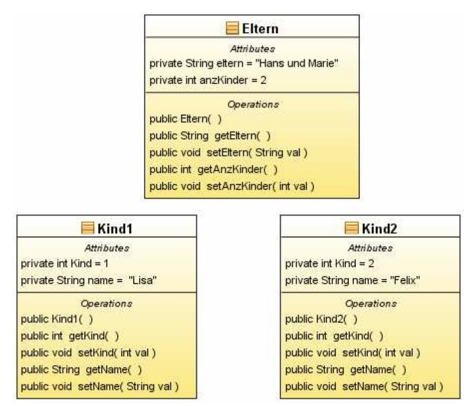


Abb. 12: Fertige Class-Elemente

Die entsprechenden get- & set-Funktionen werden automatisch von Netbeans erstellt. Nun wird die Klasse “Kind1“ mit den “Eltern“ verbunden. Dazu wird die Klasse “Kind1“ ausgewählt. Es erscheint neben der Klasse die verfügbaren Verbindungsmöglichkeiten. Hier verwenden Sie “Generalazisation“ und ziehen den Pfeil auf die Klasse “Eltern“. Da Sie “Kind1“ von den “Eltern“ ableiten möchten und die Klasse “Eltern“ Funktionen besitzt, erscheint ein neues Fenster, in dem Sie gefragt werden, ob Sie Funktionen der “Eltern“ in die “Kind1“ Klasse ableiten möchten.

Zur Ihrer Veranschaulichung:

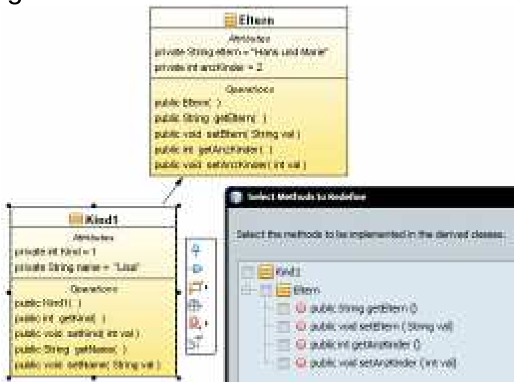


Abb. 13: Abfrage bezüglich Ableitung von Funktionen

In diesem Beispiel wurde keine Funktion gewählt. Nun wird “Kind2“, ebenso wie “Kind1“ mit den “Eltern“ verbunden:

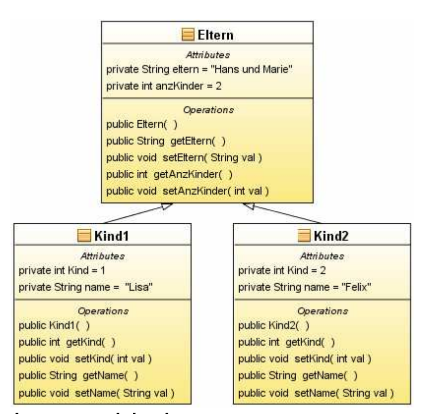


Abb. 14: Das Diagramm nach dem Verbinden

Um es etwas übersichtlicher bzw. funktioneller zu machen, fügen wir alle Elemente in einem Paket zusammen. Dafür ziehen Sie sich ein Package-Element in Ihren Arbeitsbereich und verbinden mittels “Containment“ das Package mit den “Eltern“ und den “Kindern“. Optisch ansehnlicher wird es mithilfe des “Hierarchical Layout“-Buttons:

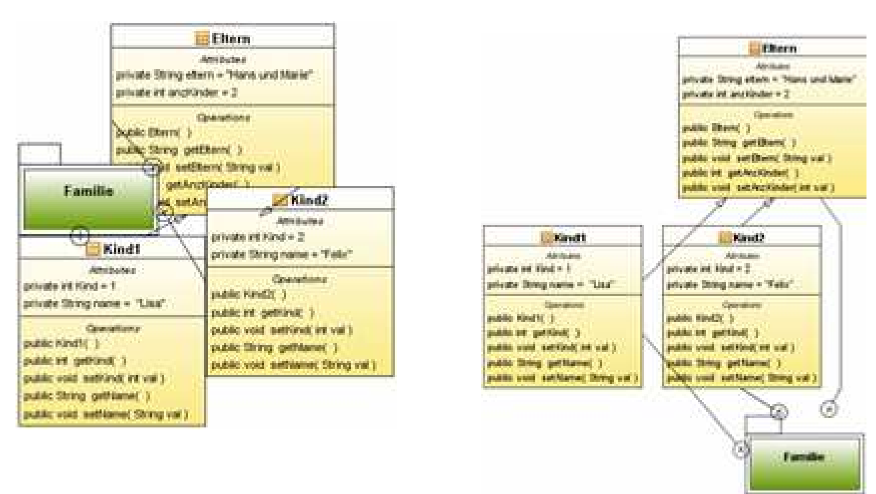


Abb. 15: Links vor und rechts nach dem hierarchischen anordnen

Forward Engineering

Aus Ihrem fertigen Klassen-Diagramm aus Kapitel 4.1 können Sie sich nun auch den entsprechenden Code von Netbeans generieren lassen.
Dazu erstellen Sie ein neues Projekt, indem der Code generiert werden soll.
Danach Rechtsklick auf das UML Projekt und die Option “Generate Code“ aus dem Kontextmenü wählen. Es erscheint folgendes Fenster:

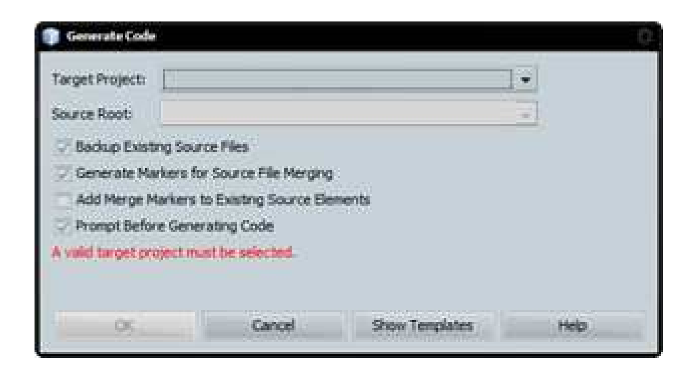


Abb. 16: Generate Code-Fenster

Hier wählen Sie unter “Target Project“ das neu erstellte Projekt aus und bestätigen mit “OK“.
Sie können natürlich auch jedes andere bereits bestehende Projekt verwenden.
Im gewählten Projekt ist nun unter “Source Packages“ unser Paket Familie (Eltern, Kind1, Kind2) zu finden. Nun kann der erzeugte Code im Projekt verwendet werden.

Zum Testen wurde folgender Main-Code verwendet:

```
package umlprojekt_class_code;
import Familie.Eltern;
import Familie.Kind1;
import Familie.Kind2;

/**
 *
 * @author Bizzi
 */
public class Main {

    /**
```

```
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
        Eltern Meier = new Eltern();
        Kind1 Tochter = new Kind1();
        Kind2 Sohn = new Kind2();
        System.out.print("Wir sind die Eltern Meier.\n");
        System.out.print("Ich bin " + Tochter.getName() + " das " +
                Tochter.getKind() + " Kind.\n");
        System.out.print("Ich bin " + Sohn.getName() + " das " +
                Sohn.getKind() + " Kind.\n");
        System.out.print("Unsere Eltern sind " + Tochter.getEltern() + " Meier.\n");

    }

}
```

Als Ergebnis und Bestätigung der Funktionalität erhalten Sie folgende Ausgabe:

```
run:
Wir sind die Eltern Meier.
Ich bin Lisa das 1 Kind.
Ich bin Felix das 2 Kind.
Unsere Eltern sind Hans und Marie Meier.
BUILD SUCCESSFUL (total time: 0 seconds)
```

Abb. 17: Programm Ausgabe

Reverse Engineering

Es besteht die Möglichkeit mit Netbeans aus einem Code ein UML Diagramm wieder zu erstellen. Zur einfachen Erklärung des „Reverse Engineers“ nutzen Sie Ihren generierten Code aus Kapitel 4.1.1.
Dazu Rechtsklick auf Ihr Projekt und “Reverse Engineer“ auswählen.
Es öffnet sich folgendes Fenster:

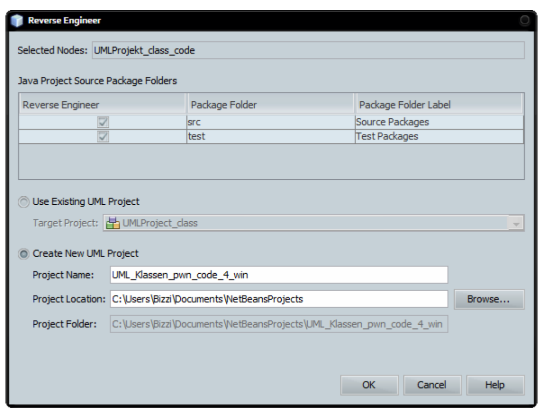


Abb. 18: Reverse Engineer-Fenster

Hier haben wir die Möglichkeit zu entscheiden, ob die Diagrammelemente in einen bestehendem Projekt oder einen neuem Projekt angelegt werden sollen. Im folgendem wurde die Option zum Erstellen eines neuen Projektes ausgewählt.
Nachdem das neue Projekt angelegt wurde,können Sie die Elemente dort wiederfinden:

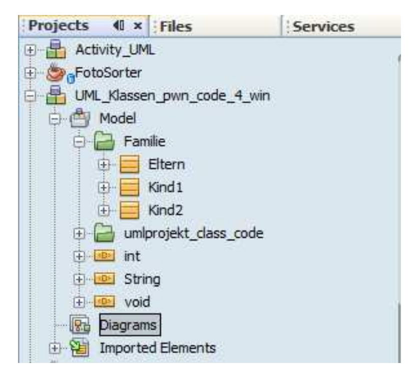


Abb. 19: Mit dem Reverse Engineer erstelltes Projekt

Allerdings ist Ihr Diagramm noch nicht im Projekt vorhanden, dies muss manuell hinzugefügt werden. Um dieses zu erhalten, Rechtsklick auf Ihr Projekt und
“New“ → “Diagram“ wählen. Danach können Sie eine beliebigen Diagrammtypen wählen. Ihre Wahl hat keinen Einfluss auf die Darstellung. Sie haben lediglich ein Problem, wenn Sie nicht wie im oben genannten Beispiel ein Klassen-Diagramm gewählt haben, da Ihnen nur die zum ausgewählten Diagrammtypen zugehörige Palette angezeigt wird und Sie somit ihr Diagramm nicht direkt mit
Klassen-Diagrammelementen erweitern können.
Nun müssen Sie nur noch die einzelnen Elemente aus dem Projektfenster in Ihr Diagramm ziehen. Alle Klassen, Funktionen, Attribute, Beziehungen etc. werden automatisch dargestellt.

So sollte Ihr Ergebnis aussehen:

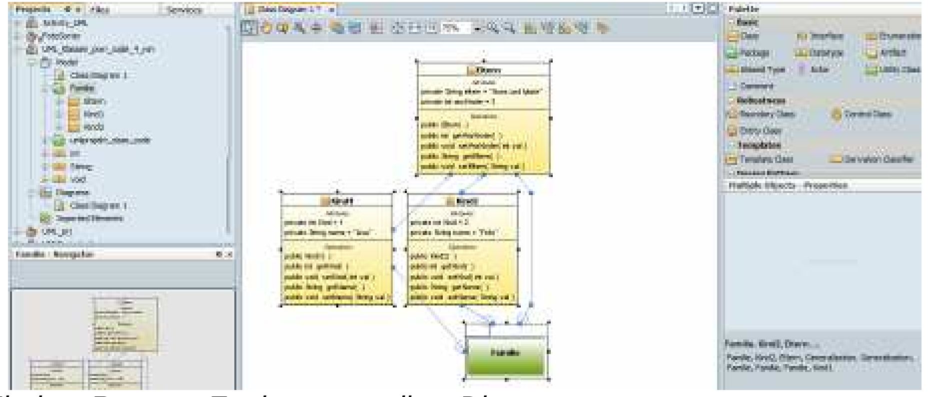


Abb. 20: Mit dem Reverse Engineer erstelltes Diagramm

4.5.2. Erstellen eines Use Case-Diagramms

Zunächst wird ein neues Projekt erstellt und als Diagrammtyp “Use Case Diagram“ ausgewählt. Danach ziehen Sie ein Aktor Element und ein Use Case Element in Ihren Arbeitsbereich. Beiden Elementen werden Namen vergeben (doppelklick Links) und eine Beziehung (Assoziation) zwischen den beiden Elementen erstellt.

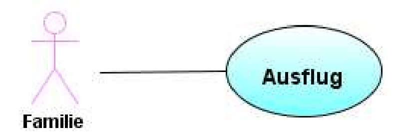


Abb. 21: Use Case Diagramm Teil 1

Nun fügen Sie weitere Use Case Elemente ein, vergeben Namen und stellen die Beziehungen untereinander her:

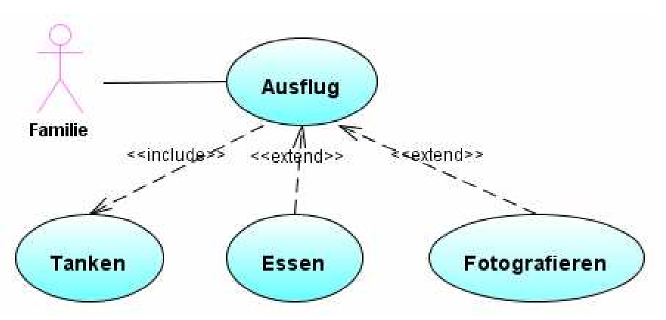


Abb. 22: Use Case Diagramm Teil 2

Um Beziehungen zu Beschriften genügt ein Linksklick auf die entsprechende Beziehung. Unter dem Kontextmenüpunkt “Labels“ finden Sie die Möglichkeit den Beziehungsnamen, die Enden und Multiplizität anzugeben.

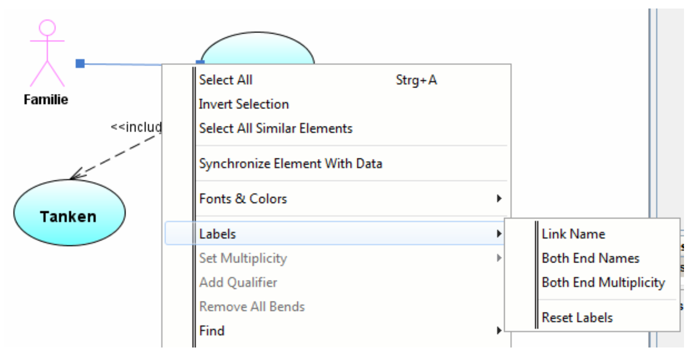


Abb. 23: Beschriftung von Beziehungen

Ihr fertiges Diagramm mit Beschriftung:

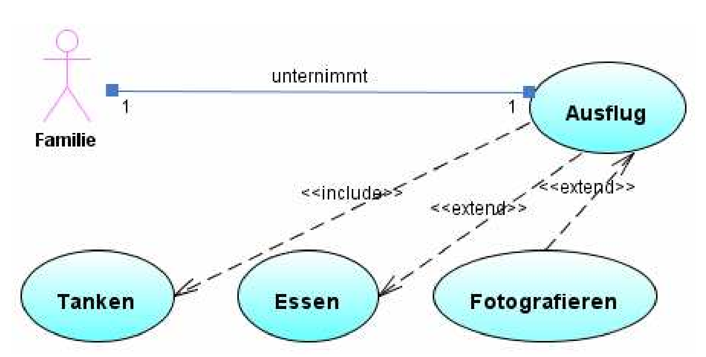


Abb. 24: Fertiges Use Case Diagramm

4.5.3. Erstellen eines Activity-Diagramms

Zunächst wird ein neues Projekt erstellt und als Diagrammtyp “Activity Diagram“ angewählt. Danach fügen Sie Ihrem Diagramm ein Partitionselement hinzu und vergeben diesem einen Namen:

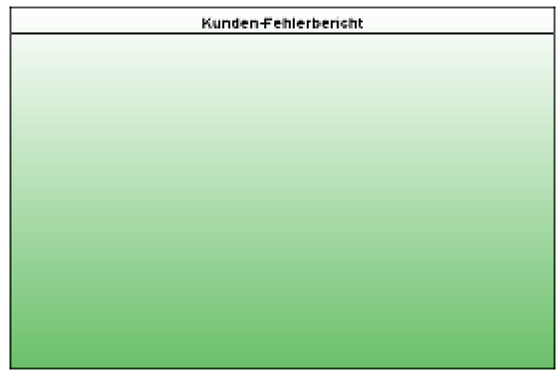


Abb. 25: Partitionselement

Die Partition teilen Sie in 2 Bereiche auf, dazu Rechtsklick in der Partition und im Kontextmenü unter “Partitions“ finden Sie die Möglichkeit Partitionen hinzuzufügen und zu löschen.

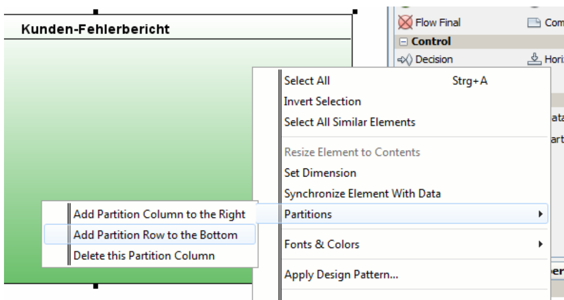


Abb. 26: Kontextmenü Partition

Es wurde eine Partition rechts hinzugefügt und den 2 Partitionen Namen vergeben. Danach wurden “Invocation“ Elemente hinzugefügt:

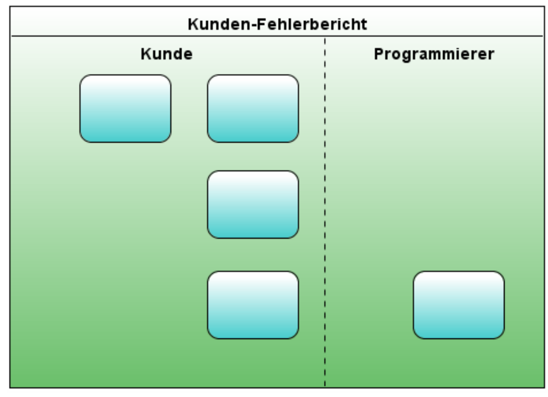


Abb. 27: Partition mit Invocation Elementen

Anschließend werden die Elemente mittels einer Bedingung miteinander verbunden. Die Verbindungen können mittels Linksklick → Labels → Show Labels angezeigt und umbenannt werden. Den “Invocation“-Elementen wurden Beschreibungen hinzugefügt:

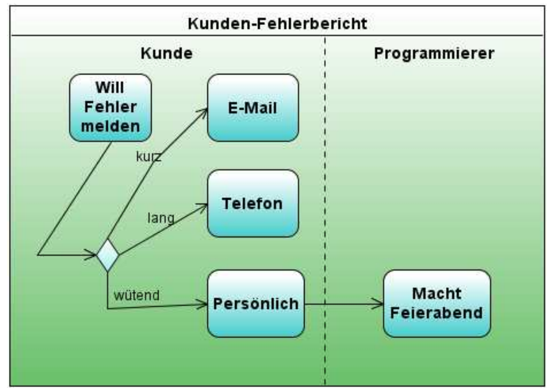


Abb. 28: Verbindungen mit Labels

Nun werden nur noch Start- und Endpunkt hinzugefügt und unser Diagramm ist fertig:

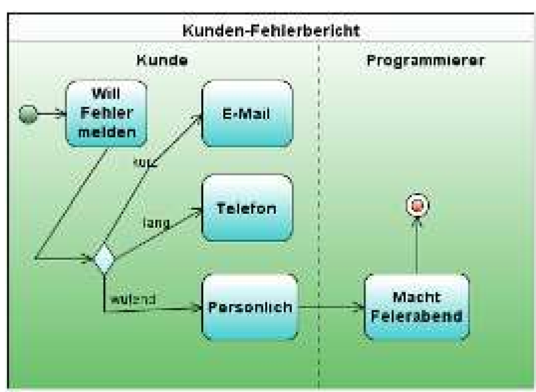


Abb. 29: Fertiges Activity Diagramm

4.6. Quellen

Netbeans IDE 6.5

5. Einführung in das Logging - Gregor Piel, 08.11.2010

5.1. Log-Datei

Eine Log-Datei enthält das automatisch geführte Protokoll aller oder bestimmter Aktionen von Prozessen auf einem Computersystem. Die korrekte Bezeichnung dafür ist deshalb Protokoll-Datei. Wichtige Anwendungen finden sich vor allem bei der Prozess-Kontrolle und Automatisierung. Während der Programm Erstellung sind Log-Dateien hilfreich um schneller Fehler zu erkennen. Auf einem PC können Protokoll-Dateien bestimmter Aktionen von einem oder mehreren Nutzern an einem Rechner geschrieben werden, ohne dass diese es bemerken oder ihre Arbeit beeinflusst wird. Das Betriebssystem hat ein eigenes System-Logbuch. Darin können u. a. die Anmeldungen am System protokolliert werden, aber auch andere wichtige Informationen. Außer dem Betriebssystem selbst schreiben meist Hintergrundprogramme (z. B. ein E-Mail-Server, ein Proxyserver und anderes) in eine Logdatei, um Aktionsmeldungen, Fehlermeldungen und Hinweise persistent (dauernd) verfügbar zu halten. Ähnliches gilt für Installations-Programme, Virenscanner und dergleichen. Logdateien werden auch von Webservern erstellt, können aber auch außerhalb des Internets bei Untersuchungen der Benutzerfreundlichkeit von Programmen oder allgemeinem Nutzerverhalten in einem System genutzt werden.

Aufbau einer Log-Datei:

- WindowsUpdate.log

  2010-09-18 16:26:07:645 964 1070
  (Zeitstempel) (PID) (TID)
  DnldMgr * Update is not allowed to download due to regulation
  (Komponente) (Beschreibung)

- Webserver

  183.121.143.32 Peter **** [18/Mar/2003:08:04:22+0200]
  (Host-Adr.) (Benutzer) (Passw.) (Zeitstempel)
  " GET /images/logo.jpg HTTP/1.1 " 200 512
  (Anforderung) (Status) (Größe der Daten)
  " http://www.wikipedia.org/ " "Mozilla/5.0 (X11; U; Linux i686; de-DE;rv:1.7.5)"
  (Internetseite) (Browser/Betriebssystem/Oberfläche)

5.2. Properties-Datei

Eine Java-Properties-Datei ist eine Textdatei, die in der Programmiersprache Java als einfacher Konfigurationsmechanismus verwendet wird. Eine *Property* (deutsch „Eigenschaft") ist in diesem Zusammenhang ein Text, der unter einem bestimmten Namen abgelegt ist. Java-Properties-Dateien haben üblicherweise die Dateiendung „`.properties`". Die Datei enthält Key/Value Paare vom Typ String. Properties sind abgeleitet von Hashtable.

Das folgende Beispiel zeigt einen Ausschnitt aus einer einfachen Konfigurationsdatei.

```
# Dies ist ein Kommentar
! Dies ist ebenfalls ein Kommentar
wenig Text
pi  = 3.14
lang:de
vieltext Dieser Text \
läuft über zwei Zeilen.
paramText = Text mit dynamischem Parameter: {0}
```

Das Beispiel enthält zwei Kommentarzeilen und definiert fünf Werte für die Namen „wenig" , „pi", „lang", „vieltext" und „paramText". Wie sich aus dem Beispiel erkennen lässt, gibt es Kommentarzeilen, die mit einem Doppelkreuz „#" oder einem Ausrufezeichen „!" beginnen, und Datenzeilen, in denen ein Name und ein Text definiert werden.

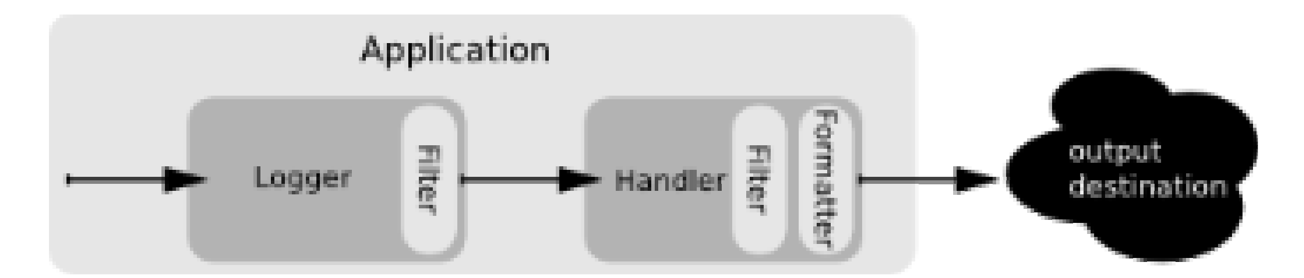
Name und Text können auf drei Arten voneinander getrennt werden, wobei die Trennzeichen nicht zum Schlüssel oder Text gehören.

1. durch ein oder mehrere Leerzeichen: Im obigen Beispiel hat der Schlüssel „wenig" den Wert „Text".
2. durch ein Gleichheitszeichen „=", umgeben von keinem oder beliebig vielen Leerzeichen: Im obigen Beispiel hat der Schlüssel „pi" den Wert „3.14".
3. durch einen Doppelpunkt „:", umgeben von keinem oder beliebig vielen Leerzeichen: Im obigen Beispiel hat der Schlüssel „lang" den Wert „de".

Ein umgekehrter Schrägstrich „\" am Ende der Zeile bedeutet, dass der Text in der nächsten Zeile weitergeht.

5.3. Java Logging Framework

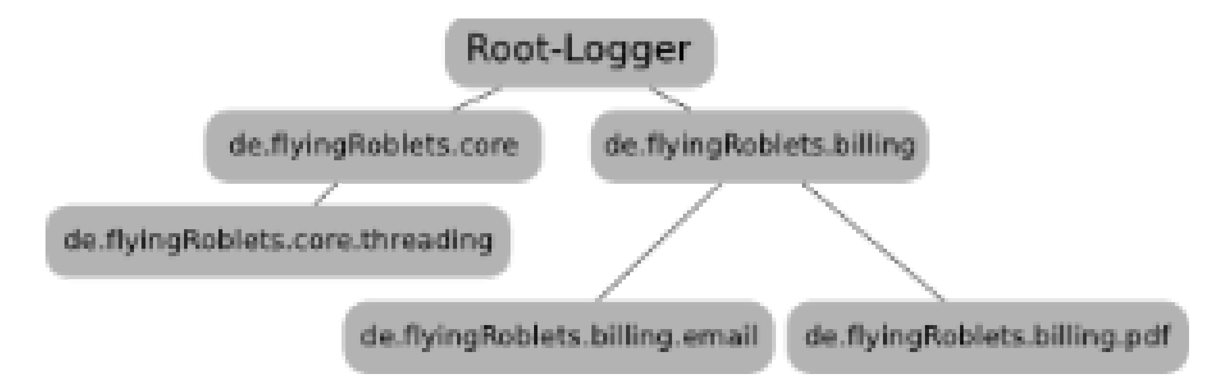
5.3.1. Logger



Der Durchlauf einer Log-Nachricht, von der Erstellung vom Entwickler im Programmcode bis zur Ausgabe z.B. auf der Konsole, beginnt mit einer Instanz der Klasse Logger. Instanzen dieser Klasse bieten Methoden zum Erstellen eines Log-Eintrages. Zu jedem Log-Eintrag wird ein Log-Level festgelegt. Dem Log-Eintrag können optional Meta-Daten hinzugefügt werden:

- Name der Java-Klasse, welche für die Erstellung des Log-Eintrages zuständig war
- Name der Methode innerhalb welcher der Log-Eintrag erstellt wurde
- Name des Ressource-Bundle zum Lokalisieren des passenden nationalsprachlichen Textes

Das Logging-Framework unterstützt den Einsatz mehrere Logger innerhalb einer Applikation. Die Logger sind untereinander in einer Baumstruktur organisiert und vererben bestimmte Eigenschaften an die jeweiligen Kinder, Kindes-Kinder usw. Die konkrete Struktur des Baumes wird durch den Namensraum definiert. Der Name eines jeden Loggers wird bei dessen Instanziierung festgelegt.



Instanziiert wird der Logger über die statische Methode getLogger(String name). Der Name wird als String übergeben. Existiert der übergebene Name schon, so wird die existierende Logger-Instanz zurückgegeben. Zum Erstellen von Log-Einträgen existieren verschiedene Methoden-Typen:

- Die "log"-Methoden
  - Als Parameter werden hier das Log-Level, die Log-Nachricht und optional beliebige Parameter übergeben.
- Die "logp"-Methoden
  - Unterscheiden sich zu den "log"-Methoden darin, dass explizit Angaben zur Methode und Klasse, in der die Log-Nachricht erzeugt wird, gemacht werden müssen.
- Die "logrb"-Methoden.
  - Unterscheiden sich zu den "logrb"-Methoden darin, dass zusätzliche Angaben zum resource bundle gemacht werden können.

Mit log<tab-Taste> bietet Netbeans die Möglichkeit eine vorgefertigte log-Methode zu benutzten in der man nur noch Details ändern muss. So fügt Netbeans folgenden Code ein:

```
Logger.getLogger(FotoFiles.class.getName()).log(Level.INFO, „Log Text“);
```

Ebenso kümmert sich Netbenas direkt um die passenden Imports:

```
import java.util.logging.Level;
import java.util.logging.Logger;
```

wird automatisch eingefügt.

Jedem Logger wird explizit ein Log-Level zugewiesen. Passiert dies nicht, wird das LogLevel des ersten Loggers übernommen, dem ein Log-Level zugewiesen wurde und in der Baumstruktur übergeordnet ist. Folgende Log-Level stehen zur Verfügung:

- Severe (höchste Priorität)
- Warning
- Info (Standard)
- Config
- Fine
- Finer
- Finest (niedrigste Priorität)

Nachdem eine der Log-Methoden aufgerufen wurde, passieren folgende Vorgänge bevor der Log-Eintrag entsteht:

1. Prüfung ob Log-Level der zu loggenden Nachricht größer oder gleich dem Log-Level des Loggers ist. Trifft dies zu, folgt Schritt zwei. Trifft dies nicht zu, wird abgebrochen und die Nachricht damit nicht geloggt.
2. Falls dem Logger Filter zugewiesen wurden, wird geprüft, ob die Log-Nachricht vom Filter erfasst wird oder nicht. Wird sie vom Filter erfasst, wird abgebrochen. Sonst folgt Schritt drei.
3. Die zu loggende Nachricht wird an alle dem Logger zugewiesenen Handler zur Weiterverarbeitung übergeben. Im Standardfall wird die Nachricht rekursiv auch an alle Handler der Vater-Logger, bis zum root-Logger, übergeben. Dieses Verhalten lässt sich mit setUseParentHandlers(boolean useParentHandlers) abstellen.

Beispiel eines Singleton-Loggers:

```
public final class Logger {
private static Logger logger;

private Logger()   {   }
      public static synchronized Logger getInstance()   {
            if ( logger == null )
                  logger = new Logger();
            return logger;
       }
      public void log( String s)   {
      System.out.println( s );
      }
}
// Logger.getInstance().log( "Log mich!" );
```

5.3.2. Handler

Sie dienen der Ausgabe von Log-Einträgen (z.b. in Datei oder Netzwerk). Bevor der Handler die Log-Einträge veröffentlicht, wird jeweils geprüft, ob das Log-Level der Nachricht größer als das des Handlers ist. Trifft dies zu, wird nachfolgend geprüft, ob mögliche gesetzte Filter den Log-Eintrag aussortieren. Filter können eingesetzt werden um Log-Einträge, vor Verarbeitung durch den entsprechenden Handler, nach beliebigen Kriterien zu filtern. Filter können auch Loggern zugewiesen werden. Besteht der jeweilige Log-Eintrag beide Tests, wird er vom Handler veröffentlicht. Es muss mindestens ein Handler existieren damit die Ausgabe funktioniert.

Beispiel eines Filters:

```
class TimeFilter implements Filter {
      int start, end;
      public TimeFilter(int start, int end) {
            this.start = start;
            this.end = end;
      }
      public boolean isLoggable(LogRecord r) {
            Calendar cal = Calendar.getInstance();
            cal.setTime(new Date(r.getMillis()));
            int h = cal.get(Calendar.HOUR_OF_DAY);
            return (start <= h) && (h <= end);
       }
 }
```

Das Java Logging Framework stellt zwei konkrete Handler bereit:

- Consol-Handler
  Die Ausgabe erfolgt in der Konsole.
- File-Handler
  Die Ausgabe erfolgt in einer Datei.

- Socket-Handler
  Die Ausgabe erfolgt im Netzwerk.

Der Handler verfügt über einen Formatter der die Log-Einträge zur Weiterverarbeitung in einen String umzuwandelt. Die Implementierung eigener Formatter ist ohne großen Aufwand möglich. Das Java Logging Framework stellt zwei Formatter zur Verfügung: Der java.util.logging.SimpleFormatter generiert menschenlesbaren Text. Der java.util.logging.XMLFormatter stellt die Log-Einträge in Form einer XML-Datei bereit.

5.3.3. Konfiguration

Grundsätzlich wird das Java Logging Framework über die Datei "lib/logging.properties" zentral für die ganze JRE konfiguriert. Diese Datei findet nur Anwendung, wenn keine der beiden Java System-Eigenschaften zur Konfiguration auf Applikations-Ebene gesetzt wurde. Sie definiert globale Einstellungen. Eigene Konfigurationen kann man entweder mit der config.class setzen oder durch eine separate config.file.

5.3.4. Konfiguration via java.util.logging.config.class

Ist diese System-Eigenschaft gesetzt, wird der Wert als Klassenname interpretiert. Diese Klasse wird dann vom Framework instanziiert. Im Konstruktor kann das Framework konfiguriert werden.

```
01 public class ConfigLogging
02 {
03
04   public ConfigLogging() throws Exception
05   {
06     // Properties erstellen
07
08     Properties loggingProperties = new Properties();
09
10     // Root-Logger Loglevel
11     loggingProperties.put( ".level", "ALL");
12     // Root-Logger Handler spezifizieren
13     loggingProperties.put( ".handlers",
14         "java.util.logging.ConsoleHandler,"
15          + "java.util.logging.FileHandler");
16
17     // Konfiguration des ConsoleHandlers
18     loggingProperties.put( "java.util.logging.ConsoleHandler.formatter",
19         "java.util.logging.SimpleFormatter");
20     // Auf der Konsole sollen nur schwerwiegende Fehler ausgegeben werden
21     loggingProperties.put( "java.util.logging.ConsoleHandler.level",
22         "SEVERE");
23
24     // Konfiguration des FileHandlers
25     loggingProperties.put( "java.util.logging.FileHandler.pattern",
26         "%h/logging/flyingroblets.log");
27     loggingProperties.put( "java.util.logging.FileHandler.limit",
28         "100 000");
29     loggingProperties.put( "java.util.logging.FileHandler.count",
30         "1");
31     loggingProperties.put( "java.util.logging.FileHandler.formatter",
32         "java.util.logging.XMLFormatter");
33     loggingProperties.put( "java.util.logging.FileHandler.level",
34         "ALL");
35
36     // Weitere Konfiguration einzelner Logger
37     loggingProperties.put( "de.flyingroblets.billing.level",
38         "INFO");
39
40     // Properties an LogManafer übergeben
41
42     PipedOutputStream pos = new PipedOutputStream();
43     PipedInputStream pis = new PipedInputStream( pos);
44
45     loggingProperties.store( pos, "");
```

```
46     pos.close();
47     LogManager.getLogManager().readConfiguration( pis);
48     pis.close();
49   }
50 }
```

5.3.5. Konfiguration via java.util.logging.config.file

Wurde die System-Eigenschaft "java.util.logging.config.class" nicht gesetzt, kann "java.util.logging.config.file" gesetzt werden um eine Konfigurationsdatei zu setzen.

```
01 ##############################################################
02 # Global properties
03 ##############################################################
04 .level = ALL
05 .handlers = java.util.logging.ConsoleHandler, java.util.logging.FileHandler
06
07 ##############################################################
08 # Handler specific properties.
09 # Describes specific configuration info for Handlers.
10 ##############################################################
11
12 java.util.logging.ConsoleHandler.formatter = java.util.logging.SimpleFormatter
13 # Auf der Konsole sollen nur schwerwiegende Fehler ausgegeben werden
14 java.util.logging.ConsoleHandler.level = SEVERE
15
16 java.util.logging.FileHandler.pattern = %h/logging/flyingroblets.log
17 java.util.logging.FileHandler.limit = 100 000
18 java.util.logging.FileHandler.count = 1
19 java.util.logging.FileHandler.formatter = java.util.logging.XMLFormatter
20 java.util.logging.FileHandler.level = ALL
21
22 ##############################################################
23 # Facility specific properties.
24 # Provides extra control for each logger.
25 ##############################################################
26 de.flyingroblets.billing.level = INFO
```

5.4. Beispiele im FotoSorter

logging.properties Datei vom FotoSorter:

```
# Specify the handlers to create in the root logger
# (all loggers are children of the root logger)
# The following creates two handlers
handlers = java.util.logging.ConsoleHandler, java.util.logging.FileHandler

# Set the default logging level for the root logger
.level = ALL

# Set the default logging level for new ConsoleHandler instances
java.util.logging.ConsoleHandler.level = WARNING

# Set the default logging level for new FileHandler instances
java.util.logging.FileHandler.formatter = java.util.logging.SimpleFormatter
java.util.logging.FileHandler.pattern = logfile.log
java.util.logging.FileHandler.level = WARNING

# Set the default formatter for new ConsoleHandler instances
java.util.logging.ConsoleHandler.formatter = java.util.logging.SimpleFormatter

# Set the default logging level for the logger named com.mycompany
# com.mycompany.level = ALL
```

Das Laden der Properties-Datei erfolgt in der main-Funktion der FotoSorter.java-Datei

```
499     public static void main(String args[]) {
500
501        System.setProperty("java.util.logging.config.file",
502                 "logging.properties");
503        try {
504             LogManager.getLogManager().readConfiguration();
505         } catch (IOException e) {
506          System.err.println(java.util.ResourceBundle.getBundle(
            "fotosort/Bundle").getString("PROBLEM TO LOAD LOGGING
            PROPERTIES"));
507         }
508         java.awt.EventQueue.invokeLater(new Runnable() {
509
510             @Override
511             public void run() {
512                 try {
513                     UIManager.setLookAndFeel(
514                             UIManager.getSystemLookAndFeelClassName());
515                 } catch (Exception e) {
516                     System.out.println("Error" + e);
517                 }
518                 new FotoSorter().setVisible(true);
519             }
520         });
521     }
```

Log-Methode in Changer.java

```
138     private int fieldToInt(JTextField jtf) {
139         int ret;
140         try {
141             ret = (Integer) Integer.parseInt(jtf.getText());
142         } catch (NumberFormatException ex) {
143             Logger.getLogger(Changer.class.getName()).log(
             Level.WARNING,"Fehler bei:" +
144                     jtf.getText()+": "+ex);
145             ret = 0;
146         }
147         return ret;
148     }
149 }
```

5.5. Quellen

- Christian Ullenboom: Java ist auch eine Insel
- Guido Krüger: Handbuch der Java-Programmierung
- Stefan Middendorf: Java Programmierhandbuch und Referenz
- http://de.wikipedia.org
- http://flyingroblets.de

6. Software Test - Max Leopold, 15.11.2010

6.1. Was ist ein Test?

6.1.1. Aufgaben von Software Test

Software Tests werden dazu eingesetzt um Fehler in Programmen zu finden. Es soll das Restrisiko für Fehler minimieren und vertrauen in den Code schaffen. Das Ziel von Software Tests ist es ein möglichst fehlerfreies Programm an den Kunden auszuliefern. Durch die Software Test kann aber nicht komplett ausgeschlossen werden, dass nicht doch Fehler existierten. Deswegen gibt es mehrere Arten von Software Tests um Fehler zu finden und zu vermeiden. Einige Testverfahren überprüfen nicht nur ob das Programm fehlerfrei läuft, sondern ob es auch der Spezifikation und der Projektplanung entspricht. Software Tests haben somit einen hohen Stellenwert um die Softwarequalität zu erhöhen.

6.1.2. Welche Arten gibt es?

- Komponententest
  Der Komponententest, auch Unit oder Modultest genannt, dient dazu nur ein bestimmtes Modul zu testen. Dabei wird nicht die Funktionalität eines ganzes Programms getestet, sondern nur eine Komponente daraus. Diese werden dann unterschiedlichen Testarten ausgesetzt.
- Integrationstest
  Der Integrationstest testet ob Module sich in das bestehende Programm eingliedern können und keinen neue Fehler verursachen.
- Systemtest
  Der Systemtest testet das ganze System. Dies geschieht beim Softwarehersteller selber. Dabei wird so gut wie möglich probiert die Bedingungen des Kunden nachzubilden. Ein Softwaretest für ein ganzes System kann teilweise Tage dauern.
- Abnahmetest
  Der Abnahmetest wird beim Kunden selber ausgeführt. Die Software wird dabei beim Kunden installiert und genutzt. Sollte der Kunde einen Fehler finden stehen direkt einige Programmierer bereit die die Fehlerhafte Komponente zu überprüfen.
- White-Box
  Beim White-Box Test ist dem Tester der Quellcode bekannt. Der Tester weiß also was das Programm macht. Das kann beim Testen Vorteile haben, da der Tester eventuell den Fehler direkt lokalisieren kann.
- Black-Box
  Beim Black-Box Test ist der Quellcode dem Entwickler nicht bekannt. Durch diese Testart können auch externe Tester eingesetzt werden, ohne Angst vor der Betriebsspionage zu haben .

In dieser Ausarbeitung werde ich auf verschiedene Arten des Komponententest eingehen. Außerdem ist der Quellcode bekannt, also handelt es sich dabei um das White-Box testen.

6.1.3. Beispiel

Ich werde in dieser Ausarbeitung anhand des Binomialkoeffizienten zeigen wie Software Tests funktionieren. Um das Programm verstehen zu können sollte die Eigenschaften des Binomialkoeffizienten verstanden sein. Mit dem Binomialkoeffizienten lassen sich die Grundaufgaben der Kombinatorik lösen.

$$\binom{\alpha}{k} = \begin{cases} \frac{\alpha(\alpha-1)(\alpha-2)\cdot \ldots \cdot(\alpha-(k-1))}{k!} & \text{wenn } k > 0 \\ 1 & \text{wenn } k = 0 \\ 0 & \text{wenn } k < 0 \end{cases}$$

Als Programm in Java sieht die Berechnung des Binomialkoeffizienten so aus:

```
public class Main {
    public static double binomialkoeffizent(int n,int k)
    {
        int i;
        double binko = 1;

        if(k>n)
        {
            binko = 0;
        }
        else if (k == 0 || k == n)
        {
            binko = 1;
        }
        else
        {
                binko = binko*(n-(k-1)+1)/(k-1);
        }

        return binko;
    }
    public static void main(String[] args) {
        int n, k;
        double binomial;
        n=3;
        k=2;

        binomial = binomialkoeffizent(n,k);
        System.out.println("Ergebnis für "+n+" über "+k+" = "+binomial);
    }

}
```

Wie wir sehen wird eine Methode zur Berechnung des Binomialkoeffizenten aufgerufen. Diese Methode hat zwei Übergabeparameter und gibt das Ergebnis der Berechnung zurück. In der Main Funktion werden drei Variablen deklariert. Zwei für n und k und eine zum abspeichern des Ergebnis. Danach wird das Ergebnis ausgegeben. Es ist ein kleines Programm aber es sollte ausreichen um die Grundlegenden Test Verfahren zu zeigen.

6.2. Statische Methoden

6.2.1. Was sind statische Methoden?

Die statischen Methoden der Software Tests haben den Vorteil, dass das zu testende Programm nicht ausgeführt werden muss. Diese Testart kann sehr früh eingesetzt werden, da kein Ausführbares Programm zur Verfügung stehen muss. Bei der Statischen Methode von Software Tests gibt es keine Testdaten oder Testfälle. Dadurch ist die statische Methode mehr Analyse als testen. Es wird dabei der Quellcode analysiert und nach einer Checkliste bewertet ob dieser Quellcode Fehler aufweißt oder etwas vergessen worden ist. Des Öfteren werden dazu mehrere Tester genommen.

6.2.2. Aufbau einer Checkliste

Hier ein Beispiel wie eine Checkliste aussehen könnte:

- Funktionsumfang/Spezifikation/Entwurf/Dokumentation
  - Ist die Funktion entsprechend der Spezifikation umgesetzt worden?
  - Ist die Dokumentation des Programms vorhanden und vollständig?
  - Enthält das Programm nicht gewünschten/spezifizieren Code?
- Programmierung allgemein
  - Gibt es mehrfach vorhandenen Code(z.B. durch mehrfaches Kopieren)
- Initialisierung und Deklaration
- Methodenaufruf
- Felder

6.2.3. Beispiel

Wie könnte so eine Checkliste in unserem Programm aussehen?

Funktionsumfang/Spezifikation

- Klein über groß wird abgefangen?
- N über 0 wird abgefangen?
- N über n wird abgefangen?
- Gibt es 2 Integer Übergabewerte?
- Heißen diese n und k?

Anhand dieser kleinen Checkliste können wir nun überprüfen ob unser Quellcode den statischen Test übersteht.

```
public class Main {
    public static double binomialkoeffizent(int n,int k)
    {
        int i;
        double binko = 1;

        if(k>n)
        {
            binko = 0;
        }
        else if (k == 0 || k == n)
        {
            binko = 1;
        }
        else
        {
                binko = binko*(n-(k-1)+1)/(k-1);
        }

        return binko;
    }
    public static void main(String[] args) {
        int n, k;
        double binomial;
        n=3;
        k=2;

        binomial = binomialkoeffizent(n,k);
        System.out.println("Ergebnis für "+n+" über "+k+" = "+binomial);
    }

}
```

In Zeile 17 sehen wir, dass klein über groß abgefangen wird. In Zeile 21 werden 2 unserer Punkte abgearbeitet. Einmal wird n über 0 und n über n abgefangen. Dann sehen wir auch das in Zeile 12 zwei Integer Übergabe Werte gibt und das diese n und k heißen.

Damit wäre unsere Checkliste abgearbeitet und der Test ist positiv verlaufen. Es hätte noch einige Punkte mehr unserer Checkliste hinzufügen können. Wir sehen das ein statischer Test sehr ausführlich und gründlich gestalten werden kann.

6.2.4. Ziele und Vorteile

Das Ziel von statischen Methoden sind logische und potenzielle Fehler zu finden. Außerdem können Verstöße gegen die Spezifikation und der Nachweis von Verletzungen der Projektplanung bewiesen werden. Da die statische Methode aufgrund ihrer Eigenschaften immer sehr früh in der Entwicklungsphase stattfindet, dauert der Test nicht sehr lange. Damit können Fehler früh erkannt und verbessert werden, was niedrige Kosten als positive Folge hat. Deshalb ist der statische Test ein Test der gerne in der Praxis genommen wird um früh zu überprüfen ob das Programm der Projektplanung entspricht. Da bei der statischen Methode oft mehrere Entwickler über den Quellcode schauen, werden öfters Verbesserungen gefunden und diskutiert. Dies wiederum trägt sehr stark zur Qualität der Software bei.

6.3. Dynamische Methoden

6.3.1. Was sind dynamische Methoden?

Die Eigenschaften von dynamischen Methoden sind fast das Gegenteil der statischen. Bei den dynamischen Software Test muss der Quellcode ausführbar sein. Das heißt das Modul muss soweit Kompilierbar und Ausführbar sein. Dadurch werden dynamische Test erst später in der Softwareentwicklung eingesetzt. Hierbei gibt es dann Testfälle und Testdaten. Unter einem Testfall versteht man einen Testrahmen mit Testdaten. Es wird bestimmt unter welchen Eingabendaten welches Ergebnis erwartet wird. Am Ende wird überprüft ob das Programm die gewünschten Ergebnisse liefert. Dabei überprüft der Test das Ergebnis selber, d.h. Der Programmierer muss nicht selber überprüfen ob das Programm korrekt gearbeitet hat. Das heißt auch das ein Test mit verschieden Parametern immer zu dem erwarteten Ergebnis führen sollte.

6.3.2. Beispiel

In dem ersten Beispiel zu Dynamischen Methoden zeige ich die Überprüfung anhand einfacher if und else Anweisungen. Dabei benutzen wir das alte Programm mit dem Binomialkoeffizienten. Der Testfall wird hierbei n = 3 u. K = 2 lauten. Das erwartete Ergebnis sollte 3.0 sein.

```
public class Main {
    public static double binomialkoeffizent(int n,int k)
    {
        int i;
        double binko = 1;

        if(k>n)
        {
            binko = 0;
        }
        else if (k == 0 || k == n)
        {
            binko = 1;
        }
        else
        {
                binko = binko*(n-(k-1)+1)/(k);
        }

        return binko;
    }
    public static void main(String[] args) {
        int n, k;
        double binomial;
        n=3;
        k=2;

        binomial = binomialkoeffizent(n,k);
        System.out.println("Ergebnis für "+n+" über "+k+" = "+binomial);
        if(binomial != 3.0)
        {
            System.out.println("Das Ergebnis ist falsch!");
        }
        else
        {
            System.out.println("Das Ergebnis ist richtig!");
        }

    }
```

Die Konsole Ausgabe:

```
Output - Referat_Beispiel1 (run)
run:
Ergebnis für 3 über 2 = 1.5
Das Ergebnis ist falsch!
BUILD SUCCESSFUL (total time: 0 seconds)
```

Wie wir sehen sagt uns das Programm das unser Test nicht erfolgreich war.
Dies zeigt uns aber eine wichtige Eigenschaft von dynamischen Tests auf. Ein dynamischer Test überprüft seine Richtigkeit selber. In unserem Programm ist also ein Fehler in der Berechnung aufgetreten. Das heißt unsere Formel zum Berechnen des Binomialkoeffizienten ist falsch. Die richtige Formel lautet also

```
else
{
        binko = binko*(n-(k-1)+1)/(k-1);
}
```

Dann verläuft der Test auch erfolgreich wie man anhand der Konsolenausgabe zu sehen ist.

Output - Referat_Beispiel1 (run)

```
run:
Ergebnis für 3 über 2 = 3.0
Das Ergebnis ist richtig!
BUILD SUCCESSFUL (total time: 0 seconds)
```

6.3.3. Nachteile

Die Nachtteile dieser einfachen Methode sind im unseren kleinen Beispiel schwer erkennbar. In größeren Projekten wird deutlich das viele solcher Kontrollanweisungen den Programm Code unübersichtlich machen. Da diese if, else und Ausgaben bei der Auslieferung an den Kunden wieder heraus genommen werden müssen wird ein weiter Nachteil deutlich. Das herraus nehmen dieser Kontrollanweisung erweist sich als schwierig, da der Unterschied zwischen einer gewollten Verzweigung und einer Kontrollanweisung schwer zu unterscheiden ist. Dafür gibt es aber eine einfache Abhilfe die Java selber mitliefert.

6.4. Assert

6.4.1. Was ist Assert?

Assert wurde entwickelt um den Programmcode zu Verifizieren. Assert stellt sicher das diese dann auch eingehalten werden. Assert geht immer davon aus das die Annahme die der Programmierer stellt wahr ist und gibt nur einen Fehler aus, wenn diese Annahme falsch ist. Assert gibt es in verschiedenen Programmiersprachen wie C/C++.

6.4.2. Syntax

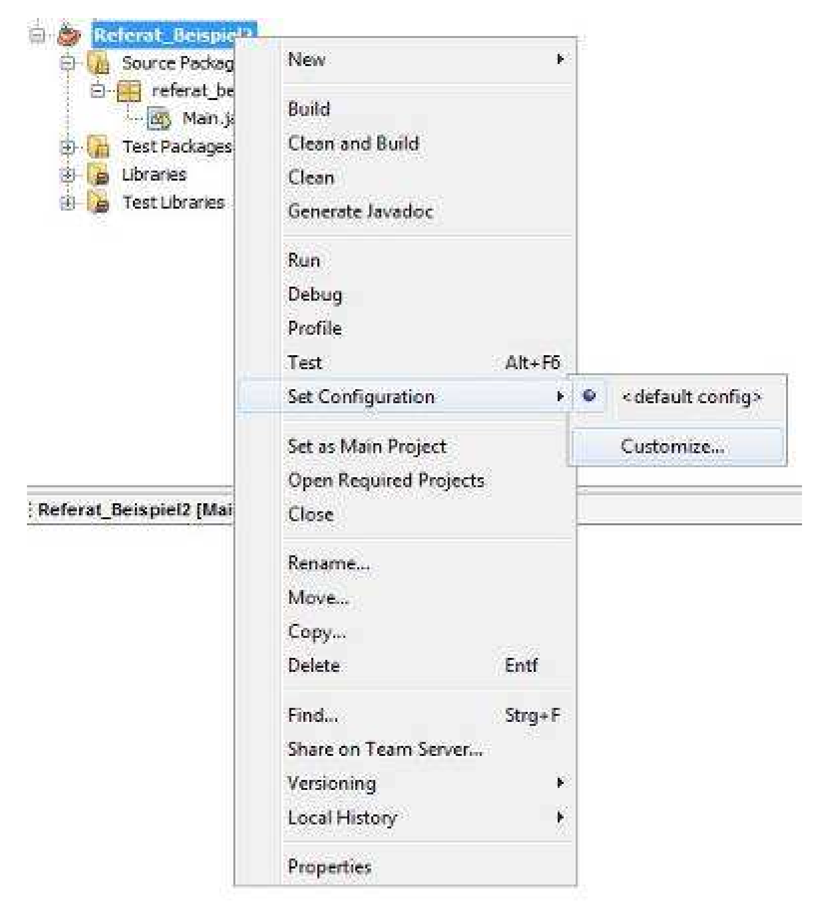
Die Syntax von Assert ist sehr einfach aufgebaut. Sie beginnt mit dem Schlüsselwort Assert und dannach mit der Annahme die getroffen wird.

```
Assert ausdruck1[:ausdruck2];
Assert x >= 0 : "x kleiner 0!";
```

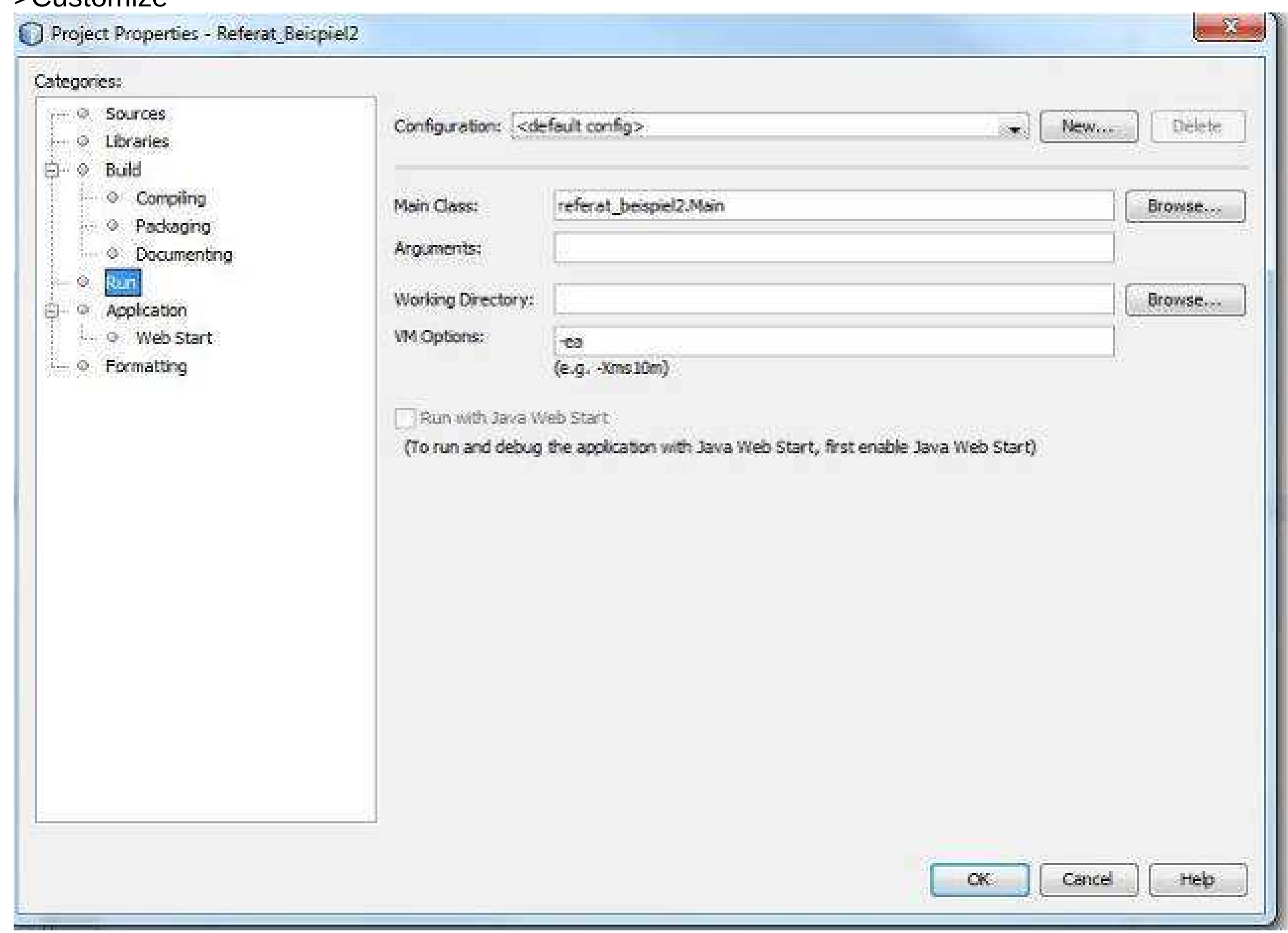
Wie wir sehen kann Assert auch einen zweiten, optionalen, Ausdruck annehmen. Dieser wird verwendet um bei einer nicht wahren Aussagen eine Exception zu werfen. Dieser wird als eine Assert.Exception geworfen. Dem entsprechend hat der Programmierer die Möglichkeit auf diese zu reagieren.

6.4.3. Beispiel

In unserem Binomialkoeffizienten behandeln wir den gleichen Testfall wie eben bei unserer if und else Methode. Es muss dabei beachtet werden, dass die Virtuelle Maschine von Java Assert ignoriert, wenn diese nicht explizit eingeschaltet ist. Dies wird durch ein Argument (-ea) an die Virtuelle Maschine übergeben.



Dafür klickt man mit der rechten Maus Taste auf sein Projekt ->Set Configuration->Customize



Dort schreibt man dann unter VM Options das zu übergebene Argument ein.

In unserem Beispiel(mit Fehlerhafter Formel) sieht dies nun so aus.

```
public static double binomialkoeffizent(int n,int k)
{
    int i;
    double binko = 1;

    if(k>n)
    {
        binko = 0;
    }
    else if (k == 0 || k == n)
    {
        binko = 1;
    }
    else
    {
        binko = binko*(n-(k-1)+1)/(k);
    }

    return binko;
}
public static void main(String[] args) {
    int n, k;
    double binomial;
    n=3;
    k=2;

    binomial = binomialkoeffizent(n,k);
    System.out.println("Ergebnis für "+n+" über "+k+" = "+binomial);

    assert binomial == 3.0 : "Ergebnis sollte 3.0 sein";


}

}
```

Wie zu sehen ist, sind die if und else Anweisungen nun verschwunden und durch eine Programmzeile ersetzt worden. Es wird immer noch überprüft ob das Ergebnis 3.0 ist. Sollte dies nicht der Fall sein, dann wird eine Exception mit dem Text "Ergebnis sollte 3.0 sein" geworfen.

```
Output - Referat_Beispiel2 (run)
run:
Ergebnis für 3 über 2 = 1.5
Exception in thread "main" java.lang.AssertionError: Ergebnis sollte 3.0 sein
        at referat_beispiel2.Main.main(Main.java:45)
Java Result: 1
BUILD SUCCESSFUL (total time: 0 seconds)
```

Ist die Formel allerdings richtig so erfolgt keine Ausgabe etc. Das Programm arbeitet sich weiter ab.

```
Output - Referat_Beispiel2 (run)
run:
Ergebnis für 3 über 2 = 3.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

6.4.4. Vorteile

Wenn wir unsere Beispiele beide dynamischen Methoden vergleichen, stellen wir fest, das der Programmcode nun wesentlich kürzer ist. Aus 5 Zeilen Code wurden 1 Zeile Code die gut sichtbar als Korrektheits-Check darsteht. Außerdem lassen sich die Assert Anweisungen explizit ein und ausschalten, so dass bei der Auslieferung der Software an den Kunden nur noch das Übergebene Argument aus der Virtuellen Maschine entfernt werden muss. Falls der Kunde später einen Fehler hat kann der Programmierer diesen leichter nachvollziehen in dem er die Tests wieder einschaltet und den Testfall durchläuft.

6.5. Junit

6.5.1. Was ist Junit?

Junit ist ein Testframework von Java Entwicklern selber. Das Aufbau Prinzip von Junit ist recht einfach, erst testen und dann Programmieren. Dafür stellt das Framework Klassen zur Verfügung, die es ermöglichen das Testmethoden schon während der Entwicklung des Programmes zu erstellen. Außerdem bietet es Klassen zur Test Protokollierung zur Verfügung. Netbeans stellt dafür einige einfache Funktionen zur Automatischen Erstellung der Testklassen. Junit ist zu dem noch Threadsicher, welches if und else sowie Assert nicht sind.Eine Ausführliche Einführung in Junit bietet das Referat und die Ausarbeitung von Marcel Höft.

6.6. Quellen

http://de.wikipedia.org/wiki/Binomialkoeffizient
http://de.wikipedia.org/wiki/Softwaretest
http://de.wikipedia.org/wiki/Statisches_Software-Testverfahren
http://de.wikipedia.org/wiki/Dynamisches_Software-Testverfahren
Handbuch der Java Programmierung (Addison-Wesley, 5.Auflage)
http://www.iks.hs-merseburg.de/~uschroet/Literatur/Java_Lit/JAVA_Insel/javainsel_08_006.htm
http://de.wikipedia.org/wiki/Assertion_(Informatik)
http://www.fh-wedel.de/~si/seminare/ws02/Ausarbeitung/6.junit/layout3.htm

7. Testen mit JUnit 4 - Marcel Höft, 15.11.2010

7.1. JUnit 4.X Neuheiten

Im JUnit 4.X gibt es einige Neuerungen gegenüber der Vorgängerversion 3.X.

- Die Testmethoden werden nicht mehr von einer TestCase- Klasse abgeleitet.
- Der Methodenname ist frei wählbar, ein Präfix ist nicht mehr nötig.
- Assert- Methoden sind nun statische Methoden der Klasse Assert
- Um Testmethoden zu identifizieren wird die @Test Annotation verwendet.

JUnit 4 nimmt dem Programmierer fast 95% der Arbeit ab. Hierzu schauen wir uns zuerst eine Klasse an und lassen uns dann von JUnit einen Test entwerfen.

7.2. Unser Beispiel ein Bank- Account

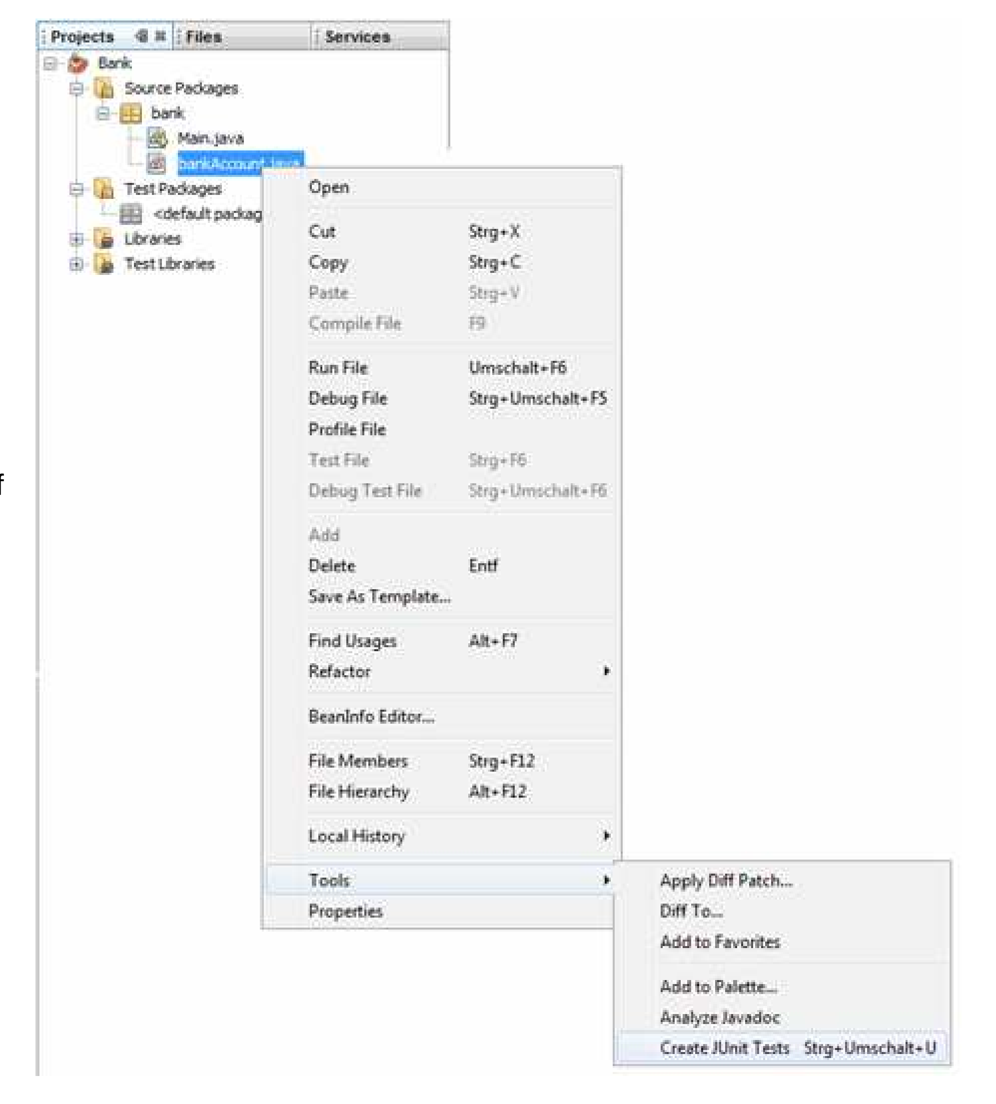
```
12   public class bankAccount {
13
14       private int geld;
15       private String besitzer;
16
17       public bankAccount() {
18       }
19
20       public bankAccount(int geld, String besitzer) {...}
24
25       public void setBesitzer(String besitzer) {...}
28
29       public void setGeld(int geld) {...}
32
33       public String getBesitzer() {...}
36
37       public int getGeld() {...}
40
41       public void einzahlen(int x) throws Exception {...}
47
48       public void abbuchen(int x) throws Exception {...}
56
57   }
```

Ein einfacher Bank- Account mit *Geld*, *Besitzer* sowie passenden Set- und Get- Methoden und die Methoden *einzahlen* und *abbuchen*.
Auf die Set- und Get- Methoden werden wir nicht weiter drauf eingehen. Interessant sind die Methoden *einzahlen* und *abbuchen*, diese sehen wir uns mal genauer an.

```
41      public void einzahlen(int x) throws Exception {
42          if(x < 0)
43              throw new Exception("Negativer Wert!");
44
45          geld += x;
46      }
47
48      public void abbuchen(int x) throws Exception {
49          if(x < 0)
50              throw new Exception("Negativer Wert!");
51          if(geld < x)
52              throw new Exception("Sie haben nicht genug Geld!");
53
54          geld -= x;
55      }
```

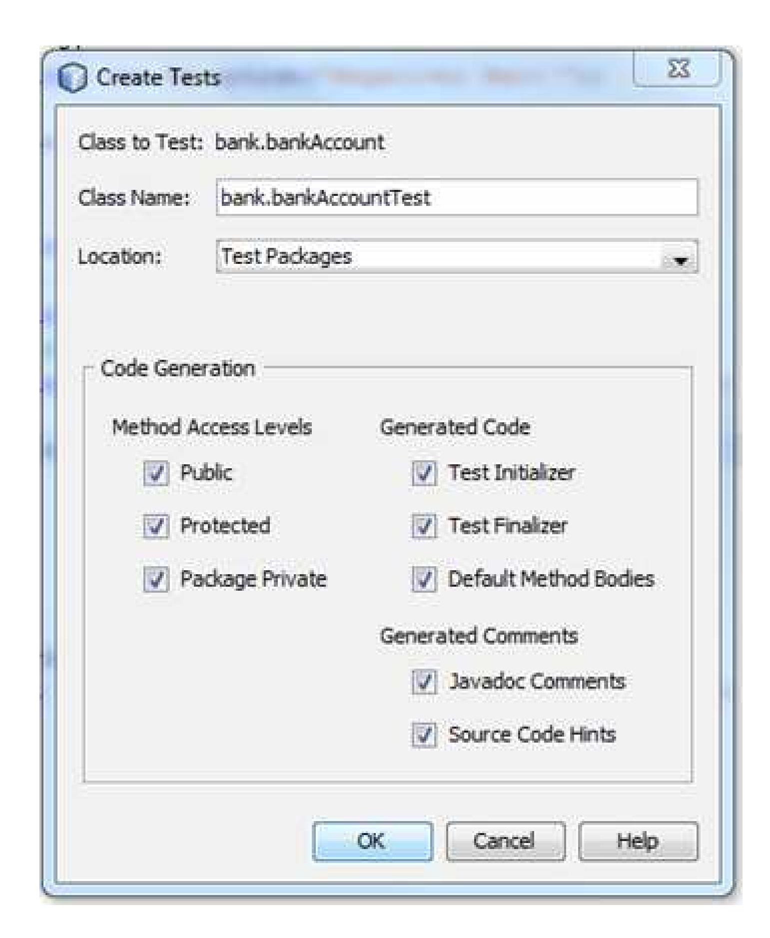
Beim Einzahlen von Geld müssen wir einen negativen Betrag abfangen und schmeißen dann eine Exception. Wenn wir Geld abbuchen wollen müssen wir ebenfalls einen negativen Betrag abfangen, aber auch prüfen ob das Konto über den abzubuchenden Betrag verfügt.
Nun soll JUnit einen Test entwerfen.

Dazu drücken wir Strg+Umschalt+U oder klicken mit der rechten Maustaste auf die .java Datei in der sich unsere Klasse befindet und klicken unter "Tools" auf "Create JUnit Tests":

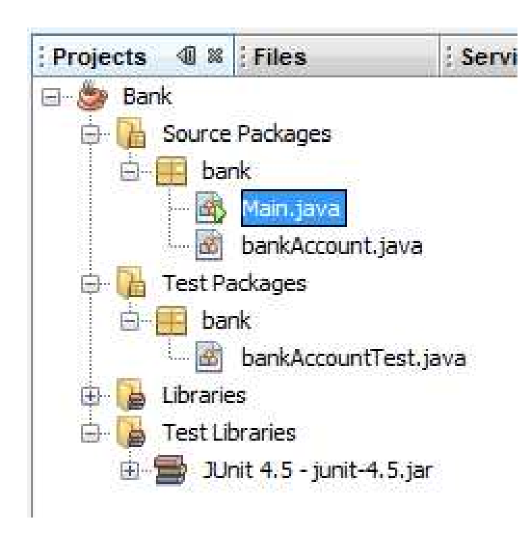


Im folgenden erscheint ein Fenster indem wir JUnit 4.X auswählen und klicken dann auf Select. Das nächste Fenster bestätigen wir mit OK.





Nun sollte unser Projekt so aussehen:



Unter "Test Packages" wurde nun ein bankAccountTest erstellt.

Den von JUnit erstellten Quellcode sehen wir uns nun etwas genauer an.

```
19  public class bankAccountTest {
20
21      public bankAccountTest() {
22      }
23
24      @BeforeClass
25      public static void setUpClass() throws Exception {
26      }
27
28      @AfterClass
29      public static void tearDownClass() throws Exception {
30      }
31
32      @Before
33      public void setUp() {
34      }
35
36      @After
37      public void tearDown() {
38      }
39
40      /**
41       * Test of setBesitzer method, of class bankAccount.
42       */
43      @Test
44      public void testSetBesitzer() {
45          System.out.println("setBesitzer");
46          String besitzer = "";
47          bankAccount instance = new bankAccount();
48          instance.setBesitzer(besitzer);
49          // TODO review the generated test code and remove the default call to fail.
50          fail("The test case is a prototype.");
51      }
52
```

Wie man sehen kann wurden diverse static Methoden erstellt. Diese lösen das frühere verwendete setUpClass, setUp usw. ab. In JUnit werden dafür nun Annotations verwendet. Es reicht vor Methoden ein @Before oder @After zu schreiben und diese Methoden werden automatisch zu Nach- oder Vorbereiter von Tests.
JUnit hat uns außerdem zu allen vorhandenen Methoden unserer Klasse bankAccount eine Testmethode geschrieben. Wir schauen uns nun unsere Interessanten zwei Methoden *einzahlen* und *abbuchen* an:

```
 94        /**
 95         * Test of einzahlen method, of class bankAccount.
 96         */
 97        @Test
 98        public void testEinzahlen() throws Exception {
 99            System.out.println("einzahlen");
100            int x = 0;
101            bankAccount instance = new bankAccount();
102            instance.einzahlen(x);
103            // TODO review the generated test code and remove the default call to fail.
104            fail("The test case is a prototype.");
105        }
106
107        /**
108         * Test of abbuchen method, of class bankAccount.
109         */
110        @Test
111        public void testAbbuchen() throws Exception {
112            System.out.println("abbuchen");
113            int x = 0;
114            bankAccount instance = new bankAccount();
115            instance.abbuchen(x);
116            // TODO review the generated test code and remove the default call to fail.
117            fail("The test case is a prototype.");
118        }
119
```

JUnit hat uns unsere angekündigten 95% bereits generiert. Durch @Test wird eine Testmethode angekündigt, die schon alles bis auf einige Assert- Befehle beinhaltet. Wir müssen nun lediglich anstelle der "fail"- Methode unseren Testfall schreiben. Beim einzahlen prüfen wir nun mit assertEquals() ob ein einzahlen von 100Euro erfolgreich war.
Beim abbuchen prüfen wir ob man 100Euro auch wieder abbuchen kann. hierbei sollten wir aber darauf achten dass auch Geld auf dem Konto vorhanden ist. Sonst würde unsere Methode eine Exception werfen.

Einzahlen:

```
 97        @Test
 98        public void testEinzahlen() throws Exception {
 99            System.out.println("einzahlen");
100            int x = 100;
101            bankAccount instance = new bankAccount();
102            instance.einzahlen(x);
103
104            assertEquals("Test einzahlen war erfolgreich", x, instance.getGeld());
105        }
106
```

```
110        @Test
111        public void testAbbuchen() throws Exception {
112            System.out.println("abbuchen");
113            int x = 100;
114            bankAccount instance = new bankAccount();
115            instance.einzahlen(x);
116            instance.abbuchen(x);
117
118            assertEquals("Test abbuchen war erfolgreich", 0 , instance.getGeld());
119        }
```

Abbuchen:

Damit sind wir schon mit den beiden Methoden fertig. Sie werde erfolgreich getestet.

testEinzahlen passed (0,001 s)
testAbbuchen passed (0,001 s)

Aber was macht man wenn man eine Falsche Eingabe Testen möchte?
Besonders beim Abbuchen sollte man testen ob auch eine Exception geworfen wird wenn das Konto nicht über genug Geld verfügt. Wir brauchen also einen Test der uns eine erwartete Exception hervorruft.
Um das zu ermöglichen kopieren wir uns einfach unseren Test vom Abbuchen und ergänzen folgende Dinge:

1. @Test(expected=Exception.class) Die Annotation @Test bekommt nun einen Übergabeparameter mit der Exception unserer Methode. Damit wird in diesem Test immer eine Exception erwartet.
2. Wir ändern den Namen der Methode, doppelte Methodennamen sind auch bei Tests nicht erlaubt.
3. Wir entfernen instance.einzahlen(x); damit das Konto beim Abbuchen leer ist.

```
122
123      @Test(expected=Exception.class)
124      public void testAbbuchenNegativ() throws Exception {
125          System.out.println("abbuchen");
126          int x = 100;
127          bankAccount instance = new bankAccount();
128          instance.abbuchen(x);
129
130
131      }
```

Und das war es auch schon. Von der Test- Methode wird nun eine Exception erwartet, die wir durch ein fehlerhaftes Abbuchen erhalten.

7.3. Die Assert-Befehle:

Durch verschiedene Assert- Befehle ist eine Vielzahl an Tests möglich. im folgenden stelle ich die wichtigsten Befehle vor.

- `assertEquals(Object expected, Object actual)`
  Vergleicht zwei Objekte:
  assertEquals("Es ist 10", 10, list.size());
  mit einer Tolleranz von 0.01:
  assertEquals("Es ist ungefähr 3 1/1", 3.33, 10.0/3.0, 0.01);

- `assertArrayEquals(Object[] expecteds, Object[] actuals)`
  Vergleicht zwei Arrays von Objekten:
  assertArrayEquals("Die Arrays sind gleich", array_B, array_B);

- `assertTrue(boolean condition)`
  Prüft ob der Vergleich richtig ist:
  assertTrue("Ist gleich", if (A == B));

- `assertFalse(boolean condition)`
  Prüft ob der Vergleich falsch ist:
  assertFalse(("Ist ungleich", if (A == B));

- `assertNull(Object object)`
  Prüft auf NULL:
  assertNULL("Es ist NULL", null);

- `assertNotNull(Object object)`
  Prüft auf Nicht NULL:
  assertNULL("Es ist nicht NULL", objekt_A);

- `assertSame(Object expected, Object actual)`
  Prüft auf die Gleichheit der Objekte. Es könnten die Selbigen Objekte sein, aber mit unterschiedlichen Inhalt:
  assertSame("Sie sind gleich", "Erwarteter Wert", "Aktueller Wert");

- `assertNotSame(Object unexpected, Object actual)`
  Prüft auf die Ungleichheit der Objekte:
  assertNotSame("Sie sind ungleich", "Erwarteter Wert", "Aktueller Wert");

- `assertThat(T actual, Matcher<T> matcher)`
  Mithilfe eines Matchers kann man hier auch auf Gleichheit prüfen:
  assertThat(list.size(), is(10));
  assertThat(list, hasItems("a", "b", "c"));

7.4. @ Annotationen

Es gibt eine Vielzahl von Annotationen unter Java und auch einige für JUnit. Einige werden wir hier nun kennen lernen.

- @Test
  Steht ein @Test vor einer Methode ist diese eine Test-Methode. Es musst sonst nichts abgeleitet werden. Es ist möglich @Test einen Übergabeparameter zu Übergeben. Zum Beispiel kann die erwartete Exception angegeben werden.
- @Test(expected=NullPointerException.class).
  Der Test läuft nur korrekt durch, wenn genau die angegebene Exception ausgeworfen wird. Bei keiner oder einer anderen Exception zeigt der Testfall einen Fehler an.

Vor- und Nachbereiter einer Methode:
- @Before
  Methoden mit @Before-Annotation werden **vor** jedem Test ausgeführt.
- @After
  Methoden mit @After-Annotation werden **nach** jedem Test ausgeführt.

Dasselbe gibt es auch für Klassen:
- @BeforeClass
- @AfterClass

Weitere:
- @Ignore
  Die Testmethode wird übersprungen
- @Timeout
  Die Testmethode wird nach einer angegebenen Zeit abgebrochen. Einsatzbereich wäre hier z. B. ein Datenbankzugriff, ist ein Verbindungsaufbau nicht möglich ist der Test nach dem Timeout Fehlgeschlagen.

7.5. Testen mit Parameterlisten

Wenn man ein größeres Programm hat, welches verschiedene Übergabeparameter akzeptiert und gesondert abhandelt, sollte man die Methode mit mehreren und

verschiedenen Werten testen. Damit wir nicht für jeden Test- Wert eine Methode schreiben müssen kann man hier eine Parameterliste zur Hilfe nehmen.
Wir haben bei unserem Beispiel leider keine große Auswahl an Test- Werten, dennoch schauen wir uns das am Beispiel der Methode abbuchen an.

Zuerst: Was ändert sich an unseren Testmethoden?

```
110     @Test
111     public void testAbbuchen() throws Exception {
112         System.out.println("abbuchen");
113         int x = 100;
114         bankAccount instance = new bankAccount();
115         instance.einzahlen(x);
116         instance.abbuchen(x);
117
118         assertEquals("Test abbuchen war erfolgreich", 0 , instance.getGeld());
119     }
```

Ohne Parameterliste:

Mit Parameterliste:

```
78      @Test
79      public void testAbbuchenOhneException() throws Exception {
80          System.out.println("abbuchen von "+testzahl);
81
82          bankAccount instance = new bankAccount(testzahl,"");
83          instance.abbuchen(testzahl);
84
85          assertTrue(0 == instance.getGeld());
86      }
```

Wie man sehen kann ändert sich nicht viel. Die verschiedenen Assert- Befehle sind hier ohne Bedeutung. Wir haben lediglich Variable "testzahl", welche in unserer Test- Klasse deklariert wird. Wir erstellen in unserer Test- Methode einen neuen Bankaccount mit dem Kontostand "testzahl" und buchen danach unseren Betrag "testzahl" wieder ab. Das kann man nun mit X- vielen Werten machen.

Nun erstellen wir uns eine Liste an Testwerten. Dazu brauche wir erst einmal folgende Packages:

```
8   import java.util.Arrays;
9   import java.util.Collection;
10  import org.junit.runners.Parameterized;
11  import org.junit.runner.RunWith;
```

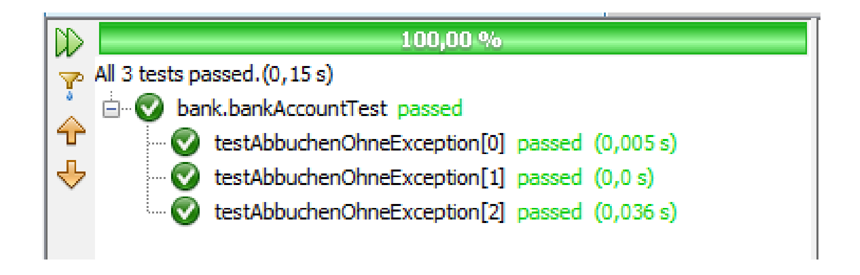
Zuletzt verändern wir unsere Test-Klasse.

```
21  @RunWith(Parameterized.class)
22  public class bankAccountTest {
23
24      private int testzahl;
25
26      public bankAccountTest(int val) {
27          this.testzahl = val;
28      }
29
30      /**
31       * Test of einzahlen method, of class bankAccount.
32       */
33
34      @Parameterized.Parameters
35       public static Collection<Object[]> data() {
36         Object[][] data = new Object[][] { { 100 }, { 200 }, { 1232542 } };
37         return Arrays.asList(data);
38       }
39
```

Wir sehen schon dass such nun einiges verändert hat, aber mehr verändert sich auch nicht bei unserem Test. Vor der Test- Klasse steht nun "@RunWith(Parameterized.class)" das unserer Klasse die Information gibt, dass wir mit einer Parameterliste arbeiten möchten.
Die angekündigte Variable "testzahl" findet man nun auch. Dazu kommt ein neuer Standard-Constructor welcher ein Übergabeparameter hat und diesen der "testzahl" zuweist.
Bis hierhin also kein Problem.

```
34      @Parameterized.Parameters
35       public static Collection<Object[]> data() {
36         Object[][] data = new Object[][] { { 100 }, { 200 }, { 1232542 } };
37         return Arrays.asList(data);
38       }
```

Schauen wir uns nun die eigentlich Test- Liste an. Wir schreiben uns eine static Methode (data) mit Rückgabewert Collection<Object[]>. In dieser Methode erstellen wir ein zweidimensionales Array des Typs "Object" und weisen ihm einige Werte zu. Wer Collection noch nicht kennt kann sich an Vectoren oder Maps orientieren, welche ähnlich arbeiten. Diesem Array übergeben wir nun unsere Testwerte mit denen wir unsere Tests durchführen wollen. Und das ist es auch schon. Wir Testen nun alle unsere Testmethoden mit jeweils den drei Werten.



7.6. Quellen

3.X
http://www.frankwestphal.de/UnitTestingmitJUnit.html
http://www2.b-topia.de/node/14
http://www.fh-wedel.de/~si/seminare/ws02/Ausarbeitung/6.junit/layout0.htm#top

4.X

http://www.cavdar.net/2008/07/21/junit-4-in-60-seconds/
http://www.ordix.de/ORDIXNews/3_2007/Java_J2EE_JEE/junit4.html
http://www.youtube.com/watch?v=8eeKvYurFU8
http://www.mkyong.com/unittest/junit-4-tutorial-6-parameterized-test/
http://www.java2s.com/Tutorial/Java/0540__JUnit/assertEqualsStringmessageexpectedactualtolerance.htm

Linkstand 23.12.2010

8. Netbeans Debugger - Tim Ingensiep, 22.11.2010

8.1. Was ist ein Debugger?

8.1.1. Allgemeine Definition

Ein Debugger ist ein Programm, was dazu dient Fehler im Quellcode zu finden und diese zu analysieren.

8.1.2. Namensherkunft

Der Name Debugger leitet sich aus dem englischen Wort *„bug"* ab. Ein „Bug" ist ein Fehler in einem Computer System. Die Bezeichnung „Bug" wurde von *Grace Hopper* geprägt, dieser fand bei einer Überprüfung eines Rechners eine Motte. Die Motte war für den Ausfall eines Relais im Rechner verantwortlich. Er klebte die Motte ins Logbuch und schrieb folgenden Satz daneben: „First actual case of bug being found." *(„Das erste Mal, dass tatsächlich ein Bug gefunden wurde.")*

8.1.3. Allgemeine Funktionen

- *Steuerung* des Programmflusses durch *Breakpoints (deutsch. Haltepunkte) und Einzelschritt-Verarbeitung von Programmbefehlen*
- *Inspizieren* von Daten. Moderne Debugger stellen Wege zu Verfügung den Inhalt eines Registers anzuzeigen. Dies kann je nach Debugger als Assembler- oder Hochsprache Output erfolgen. Daten können z.B. auch Variablen im ausführenden Programm sein. Als Beispiel will ich hier einmal eine Ausgabe eines *gdb* Debuggers zeigen der den derzeitigen Inhalt eines NSMutableArray der Programmiersprache Objective-C ausgibt. Dies geschieht mit der Hilfe des *po* Befehls.
```
(gdb) po array
<__NSArrayM 0x5f10750>(
1,
2,
3,
4,
5)
```
  Man kann an diesem Output sehr schön den Inhalt(*1,2,3,4,5*) und den Typ(*__NSArrayM*) der Variable, sowie die Speicheradresse(*0x5f10750*) sehen. Des Weiteren bieten Debugger die Möglichkeit einen sogenannten *CallStack* einzusehen. Beim *CallStack(deutsch. Aufrufliste)* handelt es sich um eine Struktur mit dessen Hilfe man nachvollziehen kann welche Methode von welcher Methode aufgerufen worden ist.
- *Modifizieren* von Daten. z.B. des Hauptspeichers, der externen Ein-/Ausgabe-Zustände und der Register des Prozessorkerns.

Je nach Debugger und Zielsystem ist es auch möglich so genannte *„Exceptions"* abzufangen (dies sind Fehler die vom Zielsystem zurück gegeben werden) In Java wird z.B. eine Exception geworfen, wenn man nicht in eine Datei schreiben kann in die man schreiben will.

Moderne Debugger unterstützen zu dem meist *„just in time debugging"* dabei muss das Programm nicht angehalten werden. Es wird zur Laufzeit neu kompiliert.

8.1.4. Debugger-Arten

Man unterscheidet grundsätzlich zwischen Debugging und Remote Debugging.

Als Debugging bezeichnet man das Debuggen eines auf dem eigenen Rechner laufenden Programms mit Hilfe von Bord-Mitteln.

Beim Remote-Debbuging läuft der Debugger auf einer anderen Maschine als das ausführende Programm. Eine andere Maschine könnte z.B. ein Server oder ein Handy sein.

Als Remote-Debugging wird ebenfalls das Debuggen von Programmen bezeichnet die in einer Simulation laufen.

8.2. Netbeans Debugger

Die Netbeans IDE enthält einen modernen Debugger der einem alles zu Verfügung stellt was man benötigt um Fehler zu finden und zu analysieren. In Verbindung mit dem *Profiler (Referat von Sascha Oedekoven)* von Netbeans ist es also sehr komfortabel sein Programm von Fehlern zu befreien und zu optimieren.
Netbeans bietet die Möglichkeit sowohl lokales als auch Remote-Debugging zu betreiben. Ebenso besteht die Möglichkeit andere Debugger für andere Programmiersprachen in die Netbeans IDE einzubinden. Es ist möglich das Debug-Verhalten, sowie die Aktion der im nächsten Abschnitt beschriebenen Debug-Knöpfe zu verändern. In diesem Skript geht es um den Java-Debugger mit standardmäßigem Verhalten.

8.2.1. Bedienung

Startet man ein Programm im Debug-Modus(*Strg+F5*) stellt einem Netbeans folgende Leiste zu Verfügung um den Debugger zu steuern:



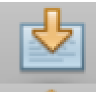
Hier eine genauere Erklärung der Buttons:



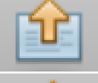
Step Over: Falls es sich bei der aktuellen Zeile um eine Methode handelt wird diese erst komplett durchlaufen und anschließend zur nächsten Zeile der aufrufenden Methode gesprungen



Step Over Expresson: Dieser Button ist dem Step Over Button ähnlich allerdings liefert er mehr Details. Wenn man diesen Button drückt, bekommt man die Eingangs und Ausgangswerte der Methoden die ausgeführt worden sind und man sieht welche Methoden innerhalb der zu überwachenden Methode aufgerufen worden sind.



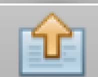
Step Into: Mit Hilfe dieses Buttons kann man in die Methode springen auf die der Debugger gerade zeigt und diese Zeile für Zeile durchgehen.



Step Out: Springt aus der derzeitigen Funktion zurück in die aufrufende Funktion, falls diese vorhanden ist.



Run To Cursor: Der Debugger hält das Programm an, wenn er an der Zeile angekommen ist, in der sich der Cursor derzeit befindet.



Apply Code Changes: Dieser Button ermöglicht es geänderten Code direkt auf das Programm anzuwenden, ohne das dieses neu gestartet werden muss. Wenn man auf diesen Button klickt, wird also das sogenannte *„just in time debugging"* ausgeführt. Diesen Button kann man erst anklicken wenn man Code geändert hat und die geänderte Datei speichert.

8.3. Breakpoints

8.3.1. Allgemein

Breakpoints(deutsch. Haltepunkte) sind Kennzeichnungen im Programm die dem Debugger mitteilen, dass er aktiv werden soll.

8.3.2. Breakpoints in Netbeans

In Netbeans gibt es sechs verschiedene Breakpoint-Typen auf die ich gleich näher eingehen werde, als erstes möchte ich allerdings erstmal Ihre Gemeinsamkeiten erläutern.
Alle Breakpoint-Typen in Netbeans haben gemeinsam, dass sie folgende Aktionen ausführen können:

- das gesamte Programm anhalten

- nur den Thread anhalten in dem sie ausgeführt werden
- das Programm einfach weiterlaufen lassen
- Text ausgeben.

Breakpoints können Bedingungen enthalten, die ihre Ausführung steuern. Diese Bedingungen können sich auf Variablen des aktuellen Kontext beziehen oder auf die Häufigkeit ihres Aufrufes(Je nach Breakpoint-Typ). Wenn man z.B. ein Indexfehler hat, der erst beim 99 Durchlauf einer Schleife passiert, kann man den Breakpoint das Programm nach dem 98 Durchlauf anhalten lassen und dann Schritt für Schritt durch gehen um den Fehler zu finden.

8.3.3. Platzhalter für Textausgabe

Bei der Textausgabe gibt es die Möglichkeit Platzhalter in den Text einzufügen die dann bei der Ausgabe ersetzt
werden. Ein paar will ich hier einmal vorstellen:

- **{=x}** : Wird durch den Wert der Variable x im aktuellen Kontext ersetzt.
- **{lineNumber}** : Wird durch die Zeilennummer in der sich der Breakpoint befindet ersetzt.
- **{className}** : Wird durch den Klassennamen ersetzt.
- **{threadName}** : Wird durch den Threadnamen ersetzt.
- **{exceptionClassName}** : Wird durch den Klassennamen der geschmissenen Exception ersetzt.
- **{exceptionMessage}** Wird durch den Text der geschmissenen Exception ersetzt.

8.3.4. Beispiel:

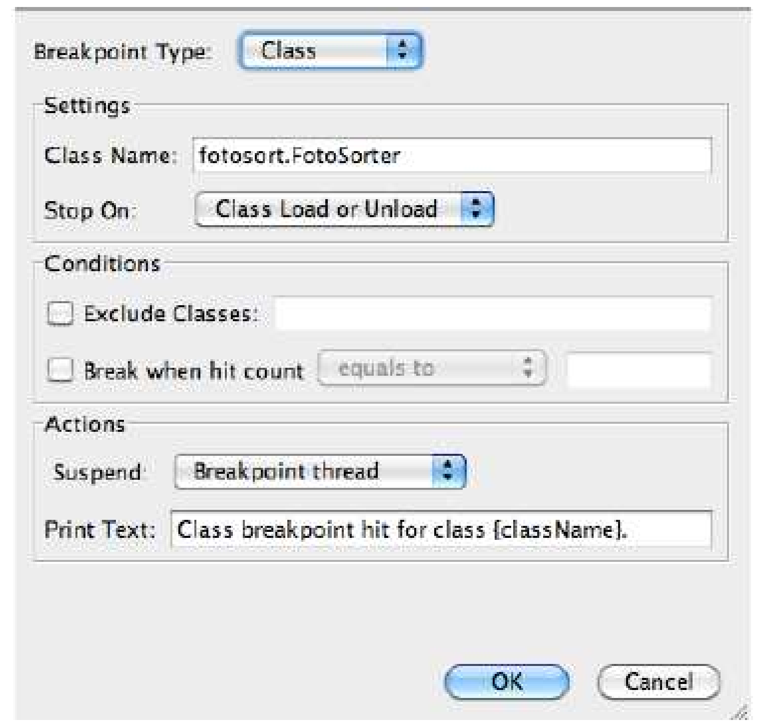
```
int sum = 0;
for (int i=0; i<3;i++) {
      sum += i;
}
```

Ein Breakpoint mit dem Print Text : *„Sum of integers 0 to {=i} = {=sum}“* würde folgende Ausgabe ergeben:

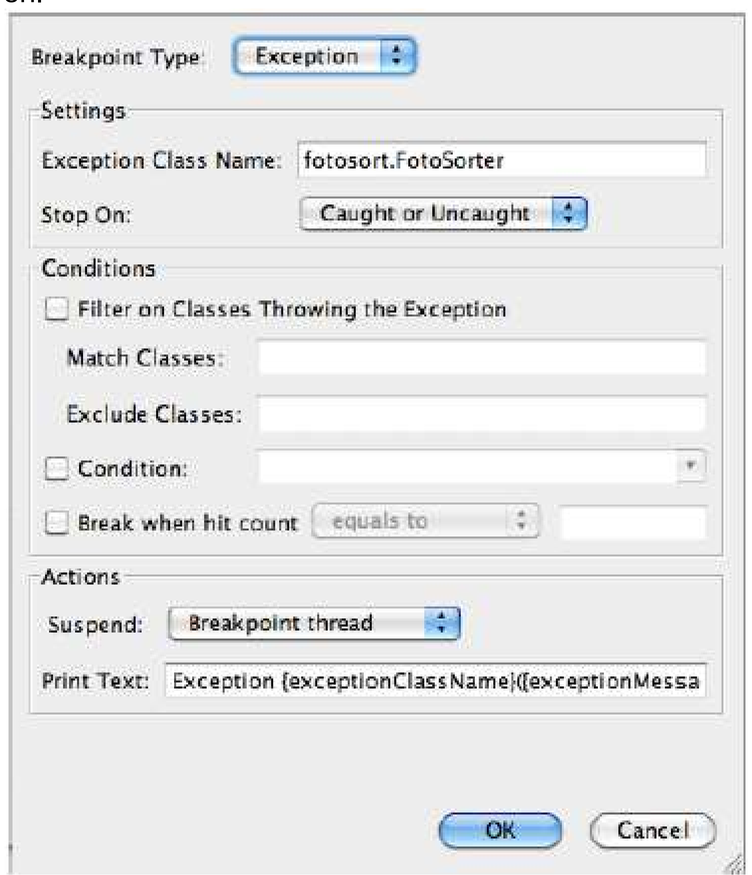
```
Sum of integers 0 to 0 = 0
Sum of integers 0 to 1 = 0
Sum of integers 0 to 2 = 1
```

8.3.5. Breakpoint-Typen

- **Class**
  Wird ausgelöst wenn eine Klasse geladen und/oder entladen wird. Man kann eine Klasse angeben auf dessen entladen und laden der Debugger reagieren soll. Es können Klassen angegeben werden auf die nicht reagiert werden soll.

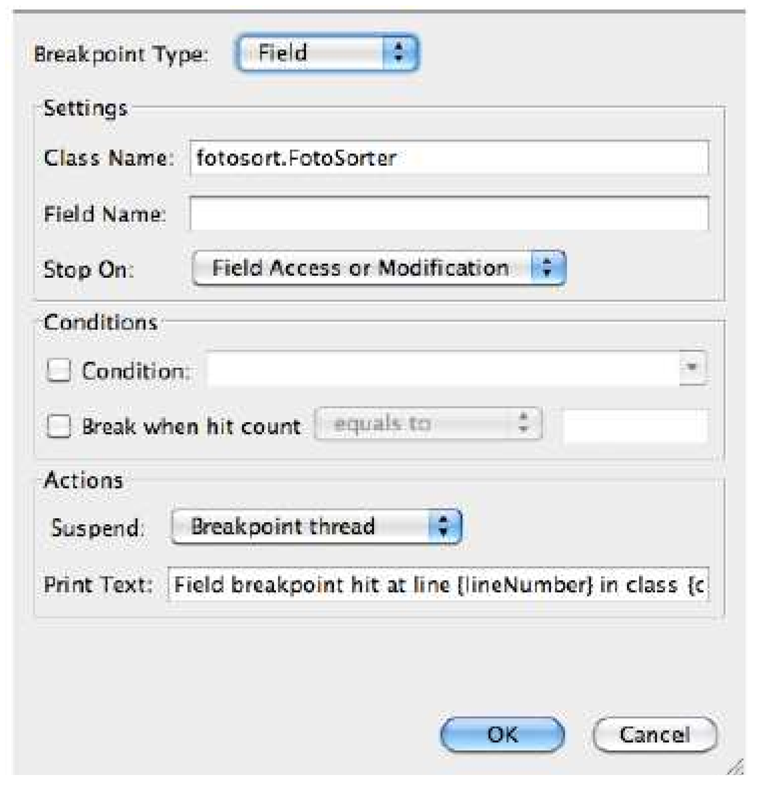


- **Exception**
  Der Breakpoint wird ausgelöst, wenn eine Exception abgefangen und/oder nicht abgefangen wird. Man kann festlegen auf welche Klassen die die Exception schmeissen der Debugger reagieren soll. Es ist eben so möglich eine Bedingung frei zu definieren.

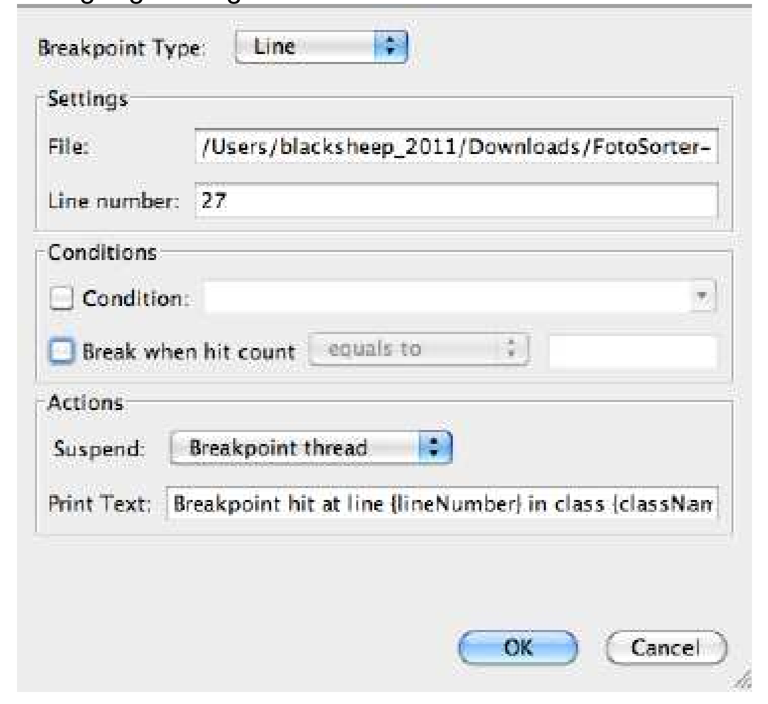


- **Field**
  Mit Hilfe dieser Breakpoint-Art wird der Debugger ausgelöst wenn auf eine Variable zugegriffen wird und/oder sie verändert wird. Es lässt sich der Variablenname, sowie die Klassen in der sich die Variable befindet definieren.Das freie definieren einer

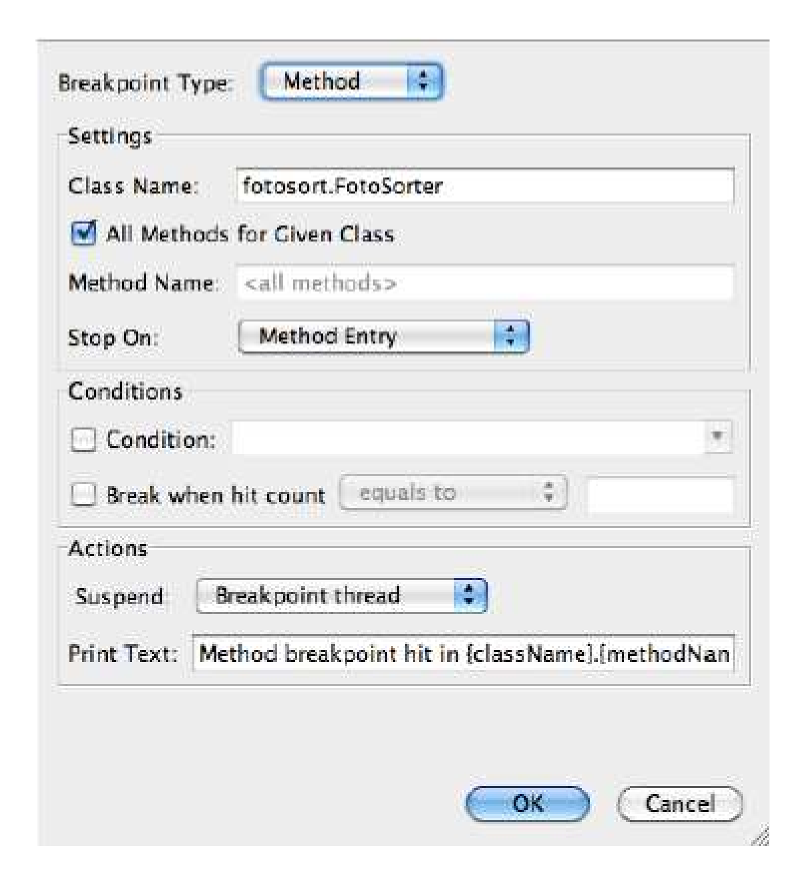
Bedingung anhand von Variablenwerten ist hier genauso möglich bei dem *Exception-Breakpoint.*



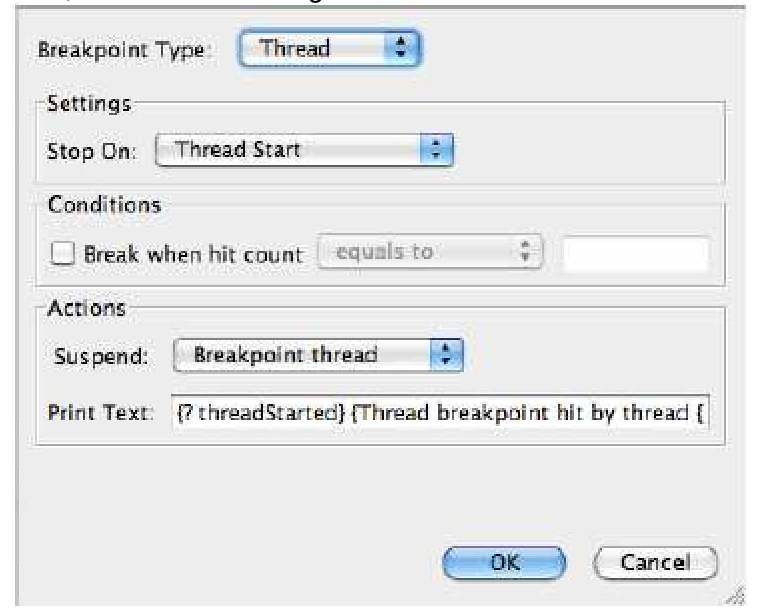
- **Line**
  Bei dieser Art führt der Debugger die Aktion aus, die in dem Breakpoint defniert ist, wenn er auf die angegebene Zeile trifft. Hier lässt sich ebenfalls wieder eine frei definierte Bedingung eintragen.



- **Method**
  Hier bezieht sich der Breakpoint auf die angegebene Methode bzw. alle Methoden der angegebenen Klasse. Der Breakpoint wird ausgelöst in den Fällen, dass die Methode betreten wird und/oder verlassen wird. Eine frei definierte Bedingung ist hier auch wieder möglich.



- **Thread**
  Wird ausgelöst, wenn ein Thread gestartet wird und/oder beendet wird.



8.3.6. Breakpoint setzen

Netbeans bietet einem verschiedenste Wege Breakpoints zusetzen. Man kann dies tun, indem man links auf die Zeilennummer klickt. Oder man kann ein Stückcode markieren und es in das Breakpoint-Fenster ziehen. In beiden Fällen ist Netbeans meistens so intelligent den passenden Breakpoint-Typ selbst zu wählen.

8.4. Watches

8.4.1. Allgemein

Mit Hilfe von Watches lassen sich der Typ und der Wert einer Variable beobachten

8.4.2. Watches in Netbeans

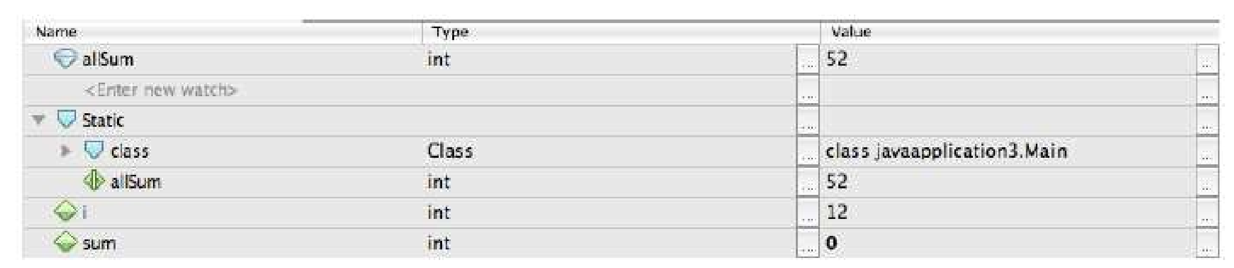
Netbeans bietet uns über *Watches* eine einfach Möglichkeit den aktuellen Wert einer Variable zu erfassen.
Damit der aktuelle Wert einer Variable angezeigt werden kann, muss sich Netbeans im Debug-Modus befinden und die Variable im aktuellen Kontext verfügbar sein.
Die lokalen Variablen der Methode die aktuell durchlaufen wird, werden automatisch in dem Watches-Fenster angezeigt. Sie haben eine grünen Route als Logo. Die manuell hinzugefügten Watches haben einen silbernen Diamanten. Wie man im unteren Screenshot sehen kann.

8.4.3. Watches hinzufügen

Es gibt folgende Wege einen *Watch* in Netbeans hinzu zufügen:

- in dem man sie per Drag & Drop auf das Watches-Fenster zieht.
- Man kann den Variablennamen im Editor markieren und im Kontextmenu *„New Watch“* auswählen.
- Eine weitere Möglichkeit besteht über das Menu von Netbeans. Unter dem Menüpunkt *„Debug“* befindet sich ein Unterpunkt *„New Watch“* über den man ebenfalls einen Watch hinzufügen kann.

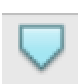
8.4.4. Watches-Fenster



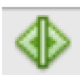
Name	Type	Value
allSum	int	52
<Enter new watch>		
Static		
class	Class	class javaapplication3.Main
allSum	int	52
i	int	12
sum	int	0



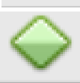
Dieses Symbol hat eine Variable oder ein Objekt das vom *Benutzer* selbst hinzugefügt wird.



Dieses Symbol haben im Watches-Fenster Einträge die mehrere einzelne Einträge zusammen fassen, weil sie nur einen Typ beschreiben. In dem obigen Beispiel z.B. der Eintrag *„Static“* unter dem alle *static* Variablen zusammen gefasst sind.



Mit diesem Symbol werden Variablen und Typen gekennzeichnet, die *static* sind.



Dieses Symbol haben alle Variablen die nicht *static* sind und im aktuellen Kontext verfügbar sind.

8.5. Evaluate Expressions

8.5.1. Allgemein

Programmcode kann ausgeführt werden, wenn das Programm vom Debugger gestoppt wird. Dieser Programmcode kann den Wert von Variablen, die im aktuellen Kontext zu Verfügung stehen, verändern.

8.5.2. Evaluate Expressions in Netbeans

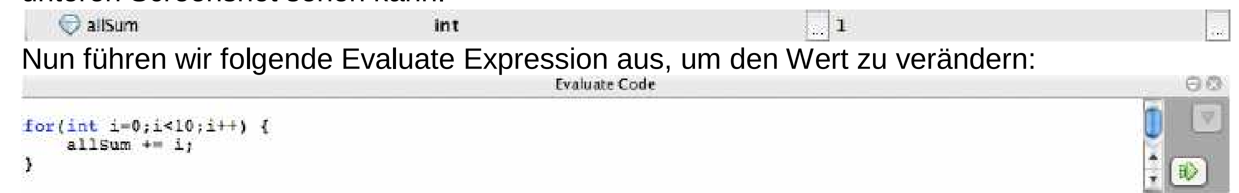
Netbeans bietet uns eine einfache Möglichkeit *Evaluate Expressions* auszuführen. Dazu klickt man im Netbeansmenü auf *„Debug“* , im dann erscheinenden Untermenü klickt man auf *„Evaluate Expression“.* Es erscheint nun ein Fenster in dem man seinen Code eingeben kann.

Beispiel:

Im folgenden Beispiel werden wir die Variable *„allSum"* mit Hilfe einer *Evaluate Expression* ändern.
Der Wert der Variable *allSum* beträgt zum Zeitpunkt des Programmstops 1 wie man am unteren Screenshot sehen kann.

allSum	int	1

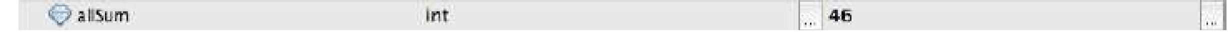
Nun führen wir folgende Evaluate Expression aus, um den Wert zu verändern:



Wenn wir unten rechts auf den grünen Pfeil klicken, wir die Expression ausgeführt, dass Watches-Fenster sieht nach der Ausführung wie folgt aus:



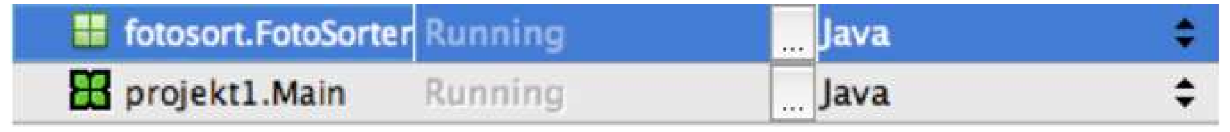
Wie man im obigen Screenshot sieht, wird das Ergebnis der Expression nach der Ausführung im Watches- Fenster angezeigt. *allSum* hat immer noch den Wert *1*. Der Wert wird erst verändert, wenn man mit dem Debugger weiter geht. Nach dem man einen Schritt weiter gegangen ist, sieht das Watches-Fenster so aus:



8.6. Multi-Session Debugging

8.6.1. Multi-Session Debugging in Netbeans

Mit Netbeans haben wir die Möglichkeit mehrere Anwendungen gleichzeitig zu debuggen. Die Ausgaben und Einstellungen des Debuggers beziehen sich immer auf die derzeitige Session. Die laufenden Sessions stellt Netbeans im Session-Fenster dar. Man kann die derzeitige Session wählen indem man im Session-Fenster rechts auf einen Eintrag klickt und im erscheinenden Kontextmenu *„Set Current Session"* auswählt. Der untere Screenshot zeigt das Session-Fenster mit zwei laufenden Sessions.



8.6.2. Session-Fenster

Die erste Spalte der Tabelle zeigt den Namen des laufenden Projektes an, durch die Icons vor den Projektnamen lässt sich erkennen welche die derzeitige Session ist.

Dieses Symbol kennzeichnet die aktuelle Session.



Durch dieses Symbol werden andere aktive Session gekennzeichnet, die aber nicht als aktuelleSession ausgewählt sind.

Die zweite Spalte stellt den Status dar. Es gibt die Stadien *„Running"* und *„Stopped"*.
In der dritten Spalte wird die in dieser Session verwendete Programmiersprache dargestellt. Es ist mit Netbeans möglich z.B. eine Session mit einem Java-Programm und eine andere Session mit einem PHPProgramm zu debuggen.

8.7. Multi-Thread Debugging

8.7.1. Multi-Thread Debugging in Netbeans

Netbeans bietet die Möglichkeit Programme mit mehreren Threads zu debuggen. Dabei kann man beim Debuggen in einen Thread gehen, nur diesen Thread anhalten oder auch alle Threads. Wo bei es natürlich zu Problemen führen kann, wenn nur ein Thread blockiert wird, je nachdem wie das Programm programmiert ist.
Die verschiedenen verschiedenen Threads des laufenden Programms sieht man im Thread-Fenster, indem man auch einige Aktionen mit den Threads durchführen kann.

8.7.2. Thread-Fenster



Der obige Screenshot zeigt das Thread-Fenster in Netbeans. Es hat eine baumartige Struktur, durch die man sehen kann, welcher Thread von welchem Thread aufgerufen worden ist. Hauptthread ist immer der *„system“* Thread.
In der linken Spalte der Tabelle steht der Name des Threads. Der dick geschriebene Name kennzeichnet den der zeitigen Thread, in diesem Screenshot wäre dies also der Thread *„main“.* Vor den Namen befinden sich kleine Symbole die ich hier genauer erklären will:

kennzeichnet die Threadgruppe, die den aktuellen Thread enthält

kennzeichnet die Threadgruppe, die den aktuellen Thread nicht enthält.

kennzeichnet den aktuellen Thread.

kennzeichnet einen laufenden Thread, der allerdings nicht der aktuelle Thread ist.

kennzeichnet einen angehaltenen Thread, der nicht der aktuelle Thread ist.

Die Rechte Spalte stellt den Status des Threads dar. Folgende Stadien sind möglich:

- **Monitor** : Thread wartet auf einen Java Monitor.
- **Not Started** : Der Thread wurde bis jetzt noch nicht gestartet.
- **Running** : Thread läuft zur Zeit
- **Sleeping** : Der Thread schläft. *Thread.sleep()* wurde aufgerufen.
- **Unknown** : Der aktuelle Status des Threads ist nicht bekannt.
- **Wait** : Der Thread wartet. *Object.wait()* wurde aufgerufen.
- **Zombie** : Der Thread hat seine Ausführung beendet, weil er fertig war.

Durch einen Rechtsklick auf einen Threadnamen öffnet sich ein Kontextmenu in dem folgende Funktionen zu Verfügung stehen:

- **Make Current** : Macht den ausgewählten Thread zum aktuellen Thread
- **Suspend** : Hält den ausgewählten Thread an
- **Interrupt** : Unterbricht den ausgewählten Thread
- **Go To Source** : Geht zum Quellcode des ausgewählten Threads

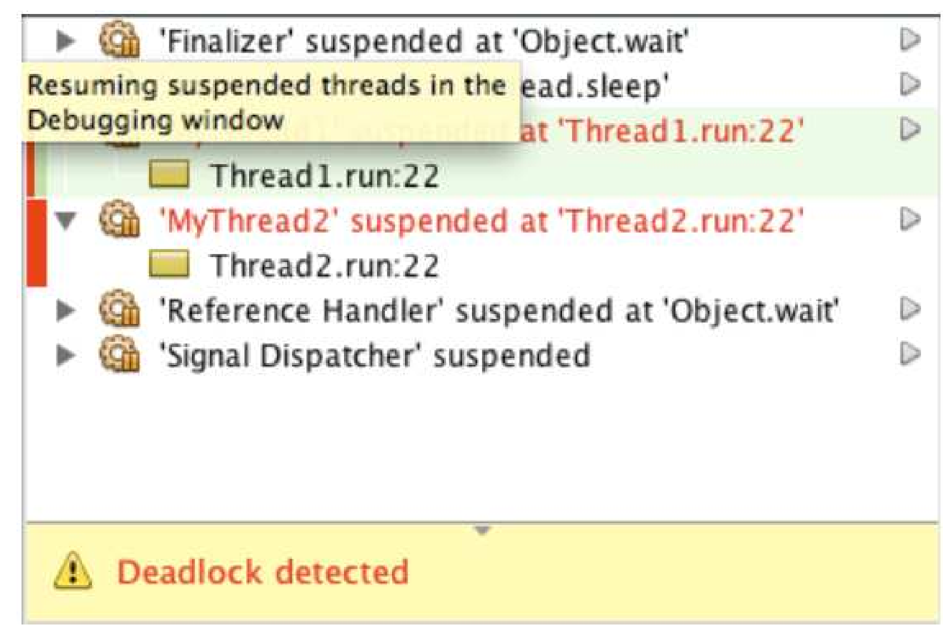
8.8. Deathlock Detection

8.8.1. Allgemeine Definition von Deathlock

Ein System wird blockiert, weil versucht wird auf eine Komponente zu zugreifen die exklusiven Zugriff voraussetzt und diese Komponente nicht wieder frei gegeben wird.

8.8.2. Deathlock Detection in Netbeans

Netbeans bietet eine Option die automatisch *Deathlocks* erkennt. Dazu geht man in Netbeans auf den Menüpunkt *„Debug“* und dann auf den Unterpunkt *„Check for Deathlock“.* Falls ein Deathlock gefunden wird, wird dies im Debugging-Fenster folgendermaßen angezeigt:



Man sieht in diesem Screenshot, dass ein Deathlock zwischen *Thread1* und *Thread2* besteht.

8.9. Tutorial

Unter folgender URL befindet sich ein Tutorial zum Thema Multi-Thread Debugging und Deathlock Detection:
Debugging Multithreaded Applications in NetBeans IDE
Unter diesem Link findet ihr eine bebilderte Schritt für Schritt Anleitung, sowie das Beispiel Projekt. Ebenfalls auf der Seite verfügbar sind Videos in denen das Tutorial abgearbeitet wird.

8.10. Quellen

- Netbeans Debugger Short Tutorial
- Java Tips - Use Netbeans Debugger breakpoints for tracing
- NetBeans IDE - Debugger Features
- NetBeans Debugger Demos: Wiki: Home — Project Kenai
- Netbeans Debugger Tutorial | eHow.com
- CSE1IOO: NetBeans Debugger
- www.javapassion.com/javase/remoteDebugging.pdf
- Debugger – Wikipedia
- Debugging Multithreaded Applications in NetBeans IDE

9. Netbeans Profiler - Sascha Oedekoven, 22.11.2010

9.1. Was ist ein Profiler?

Mit Hilfe eines Profilers können langsame Methoden und Memory-Leaks identifiziert werden. Allgemein dient er zur Laufzeitverhalten Analyse, Speicherüberwachung und zur Überwachung von Threads. Alles hat irgendwo Grenzen, so auch das Potenzial des Profilers. Zum Beispiel entstehen in Informationssystemen zu 80% die Probleme aus Datenbankzugriffen. Zudem ist der Profiler kein Wunderwerkzeug, man muss die Daten, die er liefert richtig Interpretieren können, hierzu ist Erfahrung nötig.

9.2. Technische Details

9.2.1. Traditional Profiler Techlnologies

Bytecode Instrumentation
Dies ist die meist verbreitete Technik. Hierbei werden Bytecodes in jede Klasse eingefügt. Das führt allerdings zu einigen Problemen. Eines ist die Flexibilität, wenn die Bytecodes einmal eingefügt wurden, können sie nicht zur Laufzeit geändert werden. Es kann entweder die ganze Anwendung oder nur ein Teil geprüft werden. Wenn die ganze Anwendung geprüft wird, kann dies zu erheblichen Schwierigkeiten durch Performance Probleme führen. Wenn allerdings nur ein Teil der Anwendung geprüft wird, dann wird möglicherweise die Fehlerstelle nicht geprüft und ein Fehler bleibt unentdeckt. Sicher können wir nun Stück für Stück die Anwendung prüfen, doch es müsste nach jedem Durchgang neugestartet werden.

Monitor Events
JVMPI und JVMTI generieren Events für Methoden-Eintritt und Methoden-Austritt, sowie Objektreservierung uvm. Mit dieser Methode kann man die VM nicht optimieren, zudem bietet es kleine Flexibilität, da es feste Events sind. Monitor Events kommen kaum zum Einsatz, außer zur Kontrolle des Garbage-Collectors.

Sampling
Es werden in einem bestimmten Zeitlichen Abstand (10ms – 100ms) Informationen vom Stack geholt und abgespeichert. Nun wird überprüft wie oft eine bestimmte Funktion bereits oben auf dem Stack liegt. Liegt sie Beispielsweise 4 mal auf dem Stack und es wird alle 10ms überprüft, dann wird gesagt, dass diese Funktion bereits 40 ms ausgeführt wird. Dies ist natürlich ungenau, zudem besteht keine Übersicht darüber, ob eine Funktion nun mehrmals nacheinander aufgerufen wurde oder durchgängig ausgeführt wurde. Das durchführen von Memory-Profiling ist mit dieser Technologie auch nicht möglich.

9.2.2. Dynamic Bytecode Instrumentation (DBI)

Der Bytecode wird nicht mehr statisch eingefügt, sondern kann zur Laufzeit eingefügt und entfernt werden. Daraus folgt, dass wir die Anwendung auch nicht mehr Neustarten müssen, was uns viel Zeitersparnis bringt. Es kann einfach eine Klasse/Methode ausgewählt werden, welche mit Bytecode versehen werden soll, diese wird dann beim nächsten Aufruf der Klasse/Methode geprüft.

9.3. Netbeans-Profiler Funktionen

9.3.1. Remote Profiling

Es ist möglich eine Anwendung auf einem Remote-System zu prüfen. Hierzu muss ein Remote-Packet auf dem zu prüfenden System installiert werden.

9.3.2. Attach Profiler

Es ist möglich eine Anwendung zu prüfen, die nicht aus der IDE gestartet wurde. Beide Anwendungen müssen mit der JDK6 gestartet sein, dann ist es möglich den Profiler dazuzuschalten, auch wenn das Programm bereits läuft.

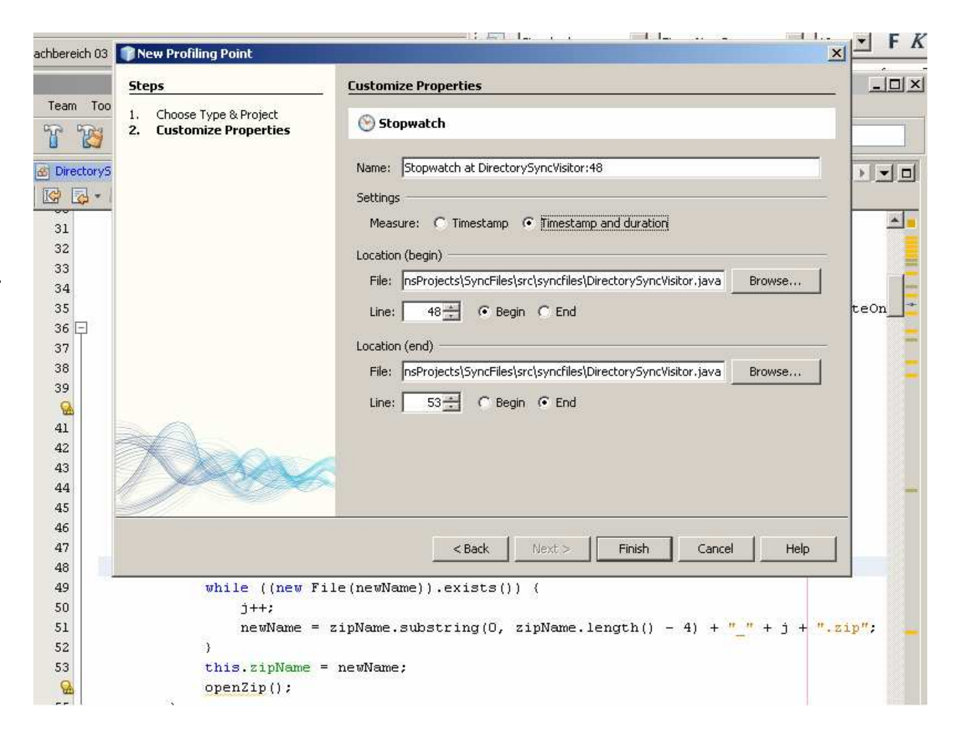
9.3.3. Profiling Points

Profiling Points sind ähnlich wie Debugger Breakpoints, sie werden ebenso im Sourcecode gesetzt. Es gibt drei verschiedene Arten:

Reset profiling results:
Alle bisher angesammelten Informationen des Profilers werden gelöscht. Dies wird hauptsächlich genutzt um an späterer Stelle ein Snapshot zu machen um diese Sequenz genauer zu analysieren.

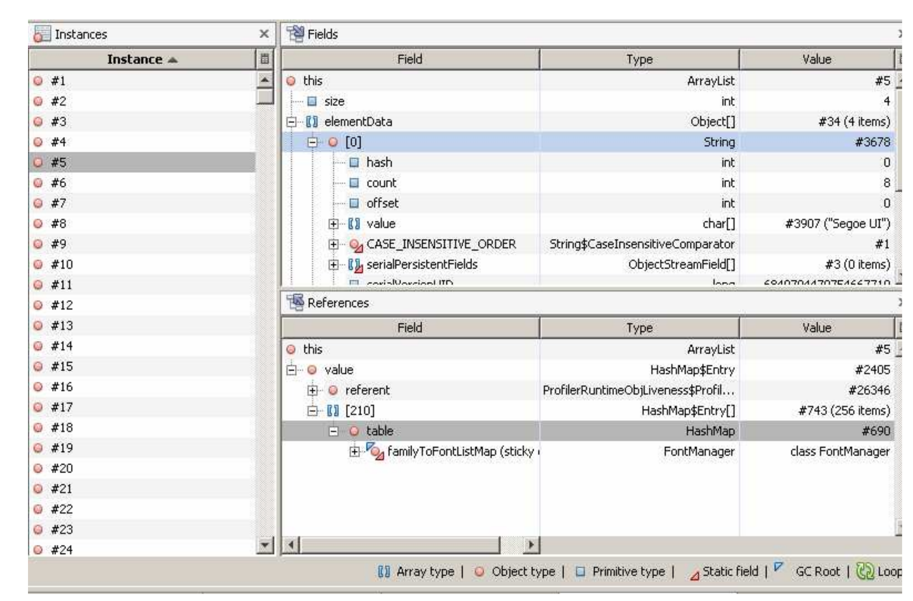
Take a snapshot :
Es wird eine detaillierte Übersicht abgespeichert, die mehr Informationen als die Live-Results liefert. Diese Snapshots kann man später auch untereinander Vergleichen.

Record the timestamp:
Wie der Name schon sagt wird hier ein Timestamp gespeichert. Zudem ist es möglich eine Dauer zu speichern. Wir können also einen Profiling Point mit Timestamp vor einer Schleife und einen nach einer Schleife platzieren. Bei durchlaufen des Programmes wird uns dann die Dauer angezeigt, wie lange er in der Schleife benötigt hat.



9.3.4. HeapWalker

Ein nützliches Tool um den Heap einzusehen und nach Memory-Leaks zu suchen. Es werden die Klassen angezeigt, die zur Zeit auf dem Heap sind. Zu jeder Klasse werden die dazugehörigen Instanzen und die Größe aller Instanzen angezeigt. Durch doppelklick auf eine der Klassen gelangen wir zu den einzelnen Instanzen der Klasse. Hier bekommen wir noch nähere Informationen zu den Referenzen.

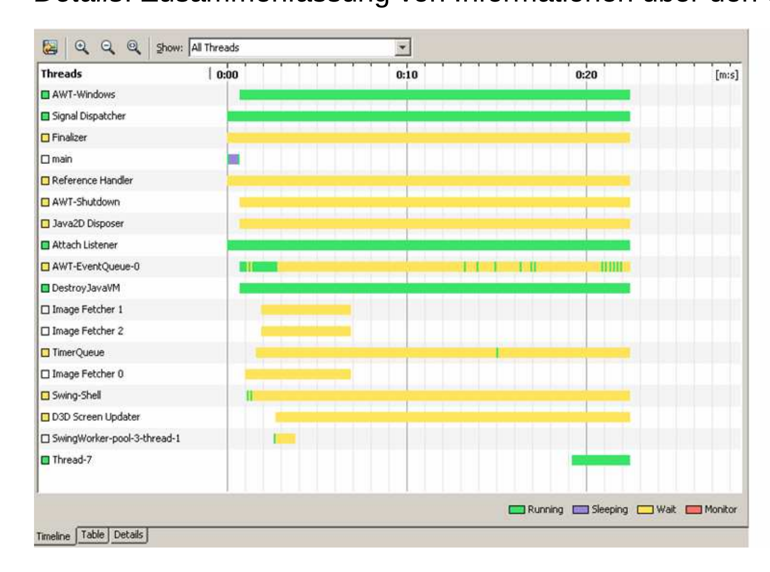


9.3.5. Monitor Application:

High level Informationen über verschiedene wichtige Eigenschaften wie z.b. Thread-aktivitäten und Speichernutzung

9.3.6. Thread Status

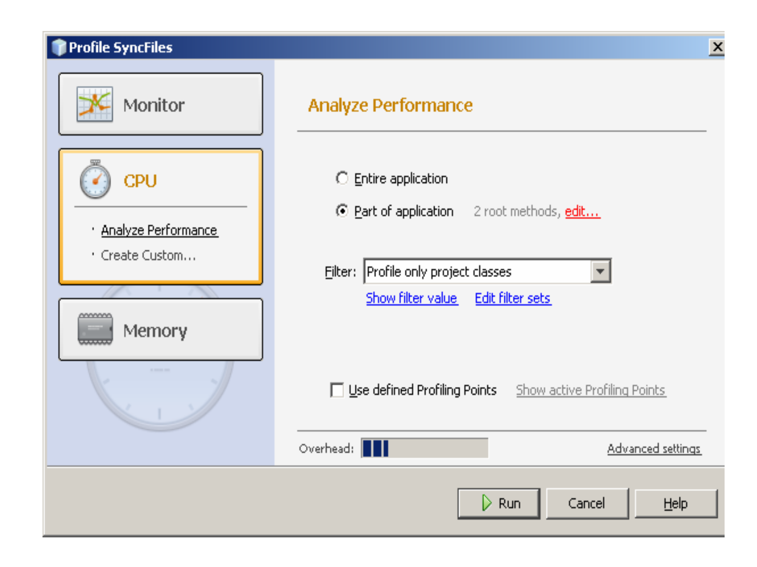
Timeline: Aktuelle und alte Thread-Status werden dargestellt. Es wird zur Laufzeit aktualisiert.
Details: Zusammenfassung von Informationen über den Status eines einzelnen Threads.



Grün: Thread läuft oder ist bereit.
Lila: Thread ist pausiert. Z.b. durch Thread.sleep()
Gelb: Thread wartet. Z.b. durch Object.wait()
Rot: Thread ist blockiert. Thread versucht in einen Synchronisierten Bereich einzutreten, muss aber warten.

9.3.7. Analyze Performance

Es wird die Ausführungszeit der einzelnen Methoden und Klassen gemessen. Dank DBI werden nur Methoden die wir prüfen wollen mit Bytecode versehen. Dadurch haben wir zum einen eine viel Übersichtlichere Ausgabe und das Programm bleibt schnell.



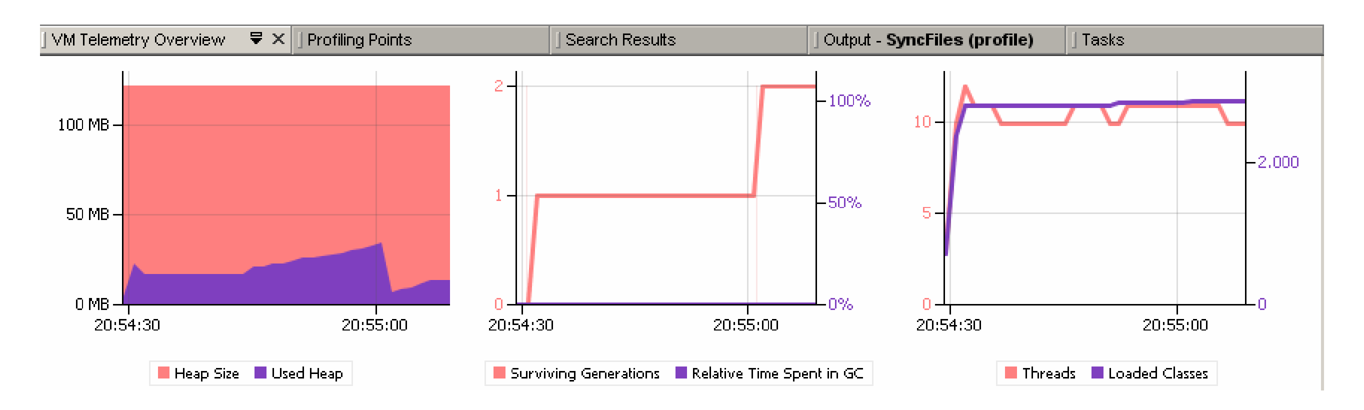
Entry Application
Alle Methoden werden überwacht. Threads senden ein „Method-enty“ Event beim eintreten in eine Methode und ein „Method-exit“ beim austreten der Methode. Beide Events haben Timestamps.

Part of Application
Es wird nichts geprüft, bis wir in eine vom User gewählte Root-Methode treffen.
Die Anwendung wird insgesamt nicht so sehr verlangsamt wie bei der Option „Entry Application“.
Einige Anwendungen müssen durch auswählen einer Root-Methode geprüft werden, da die Anzahl der generierten Daten sonst unbrauchbar wären oder die Anwendung sogar Crashen könnte. (unexpected timeouts)

Custom Profiling
Konfigurationen der Filter etc. werden hier gespeichert. Zu gebrauchen, wenn z.b. eine Webanwendung geprüft werden soll, die auf Apache Tomcat läuft, durch die Filter können hier die Tomcat eigenen Sachen ausgeschaltet werden, sodass nur die eigene Anwendung geprüft wird.

9.3.8. VM Telemetry Overview

Wird immer angezeigt, wenn wir Monitoring Application ausgewählt haben. Ansonsten kann man es über Profiler → View → Telemetry Overview dazuschalten.
Auf der Linken Seite sehen wir in rot den reservierten Speicher des Heaps. Das Lila zeigt den aktuell genutzen Speicher des Heaps an. Rechts wird die Anzahl der aktiven Threads angezeigt.
In der Mitte sehen wir zwei wichtige Heap Statistiken.
Die blaue Linie zeigt die Zeit an, die der Garbage-Collector benötigt. In dieser Zeit kann unsere Anwendung nicht weiter ausgeführt werden. Wenn dies einen zu hohen Anteil annimmt, sollte man den Heap erweitern. (-Xmx parameter)
Die Rote Linie zeigt die überlebenden Generationen. Die Anzahl der überlebenden Generationen ist die Anzahl der verschiedenen alter aller Java Objekte auf dem Heap. Wenn ein Objekt einen Garbage-Collector-Zyklus überlebt hat, wird das Alter um eins erhöht. Wenn die Anzahl der Generationen hoch ist, bzw. ständig steigt, dann werden immer wieder neue Objekte erzeugt und Speicherplatz reserviert, obwohl noch Platz für die alten Objekte reserviert ist.
Dies ist Verschwendung von Speicherplatz, also ein Memory-Leak.



9.3.9. Live Results

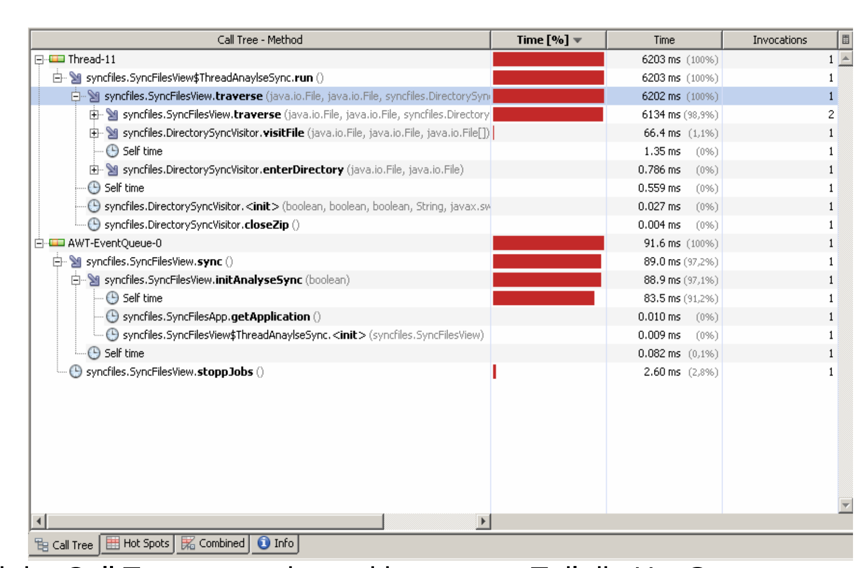
Je nachdem welche Profiling-Art wir ausgewählt haben wird nun eine Statistik über die Geschwindigkeit oder aber über die Objektreservierung und die Lebenslänge.
Diese Statistik wird automatisch alle 2 Sekunden aktualisiert.

9.3.10. CPU Snapshot

Der CPU Snapshot sammelt Daten über die Zeit und die Anzahl der Aufrufe einer Funktion.
Die folgenden Menüs sind auswählbar:

Call Tree: Ausführungszeit und Anzahl der Aufrufe einer Methode werden angezeigt. Es wird unterschieden in welchem Kontext.

Hot Spots: Gesamte Ausführungszeit und Anzahl der Aufrufe einer Methode werden angezeigt, egal in welchem Kontext sie ausgeführt wurden.

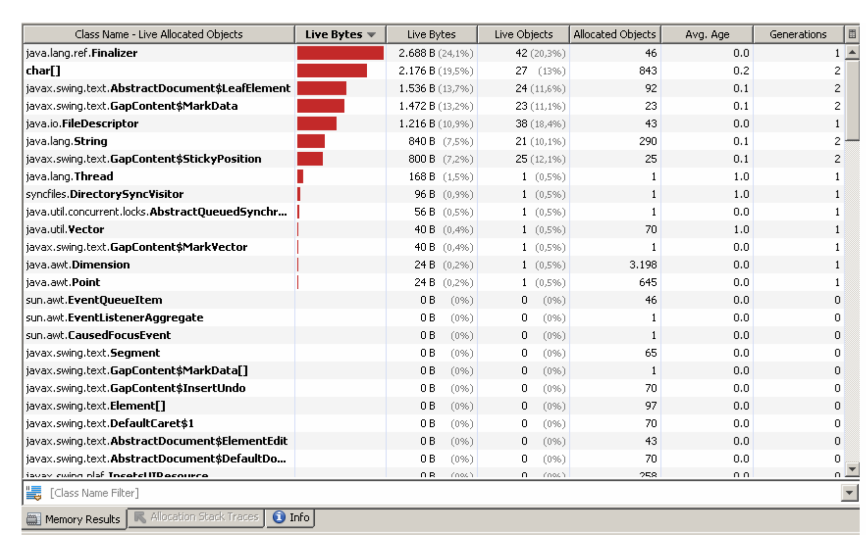


Combined: Im oberen Teil wird der Call Tree angezeigt und im unteren Teil die Hot Spots.
Info: Informationen über das Erstellungsdatum, Speicherort und Überprüfungsart.

9.3.11. Memory Snapshot

Der Memory-Snapshot sammelt Daten über die Reservierung und Lebensdauer von Objekten.
Memory Results: In diesem Fenster wird eine Liste von Klassen mit dazugehörigen Statistiken über die Größe und Anzahl der Instanzen die während des Snapshots reserviert waren angezeigt.
Allocated Objectes: Anzahl aller Objekte die der Profiler zur Zeit überwacht.
In der Regel ist dies ca. 10% von der wirklichen Anzahl aller Objekte. Dadurch, dass nicht alles überwacht wird, ist der Netbeans Profiler deutlich schneller und kann fast bei voller Geschwindigkeit arbeiten. Das Ergebnis ist jedoch kaum verändert gegenüber einer Überwachung von 100%.
Live Objects: Objekte die zur Zeit auf dem Heap sind und somit auch Speicherplatz beanspruchen.

Live Bytes: Es wird angezeigt wieviel Speicherplatz von den „Live Objects“ auf dem Heap genutzt wird.
Avg.Age: Durchschnittliche „Alter“ der live Objects. (Alter = überlebte garbage-collector-Zyklen)
Generations: Anzahl der Objekte verschiedenen alters.



Class Name - Live Allocated Objects	Live Bytes	Live Bytes	Live Objects	Allocated Objects	Avg. Age	Generations
java.lang.ref.Finalizer		2.688 B (24,1%)	42 (20,3%)	46	0.0	1
char[]		2.176 B (19,5%)	27 (13%)	843	0.2	2
javax.swing.text.AbstractDocument$LeafElement		1.536 B (13,7%)	24 (11,6%)	92	0.1	2
javax.swing.text.GapContent$MarkData		1.472 B (13,2%)	23 (11,1%)	23	0.1	2
java.io.FileDescriptor		1.216 B (10,9%)	38 (18,4%)	43	0.0	1
java.lang.String		840 B (7,5%)	21 (10,1%)	290	0.1	2
javax.swing.text.GapContent$StickyPosition		800 B (7,2%)	25 (12,1%)	25	0.1	2
java.lang.Thread		168 B (1,5%)	1 (0,5%)	1	1.0	1
syncfiles.DirectorySyncVisitor		96 B (0,9%)	1 (0,5%)	1	1.0	1
java.util.concurrent.locks.AbstractQueuedSynchr...		56 B (0,5%)	1 (0,5%)	1	0.0	1
java.util.Vector		40 B (0,4%)	1 (0,5%)	70	1.0	1
javax.swing.text.GapContent$MarkVector		40 B (0,4%)	1 (0,5%)	1	0.0	1
java.awt.Dimension		24 B (0,2%)	1 (0,5%)	3.198	0.0	1
java.awt.Point		24 B (0,2%)	1 (0,5%)	645	0.0	1
sun.awt.EventQueueItem		0 B (0%)	0 (0%)	46	0.0	0
sun.awt.EventListenerAggregate		0 B (0%)	0 (0%)	1	0.0	0
sun.awt.CausedFocusEvent		0 B (0%)	0 (0%)	1	0.0	0
javax.swing.text.Segment		0 B (0%)	0 (0%)	65	0.0	0
javax.swing.text.GapContent$MarkData[]		0 B (0%)	0 (0%)	1	0.0	0
javax.swing.text.GapContent$InsertUndo		0 B (0%)	0 (0%)	70	0.0	0
javax.swing.text.Element[]		0 B (0%)	0 (0%)	97	0.0	0
javax.swing.text.DefaultCaret$1		0 B (0%)	0 (0%)	70	0.0	0
javax.swing.text.AbstractDocument$ElementEdit		0 B (0%)	0 (0%)	43	0.0	0
javax.swing.text.AbstractDocument$DefaultDo...		0 B (0%)	0 (0%)	70	0.0	0

Allocation Stack Traces: Dieses Fenster kann aufgerufen werden, indem man in den Memory Results mit einem Rechts-klick auf eine Klasse geht und es dort auf „Show Allocation Stack Traces“ klickt. Es werden von der Klasse ausgehende Funktionen gezeigt.
Info: Informationen über das Erstellungsdatum, Speicherort und Überprüfungsart.

9.4. Beispiel – Anhand des Programmes SyncFiles

Beschreibung: In diesem Beispiel analysieren wir das Programm SyncFiles auf Threadtauglichkeit. Danach sollten Sie in der Lage sein den Profiler zu Kalibrieren und eigenständig zu nutzen. Grundlagen zu den in dem Beispiel nicht angesprochenen Punkten finden Sie in den Punkten 1-3 des Scriptes.

1. Projekt SyncFiles hier runterladen *http://home.arcor.de/ka.menzel/software/SyncFiles.zip* und in Netbeans öffnen.
2. Als erstens müssen wir den Profiler Kalibrieren. Dazu gehen Sie bitte auf den Menüpunkt *Profile → Advanced Commands → Run Profiler Calibration*.
3. Nun ist der Profiler einsatzbereit. Wir werden unser Projekt auf Memory und Thread Leaks untersuchen. Hierzu wählen Sie in dem Menü *Profile* den Unterpunkt *Profile Main Projekt* aus. In dem nun erscheinenden Fenster wählen Sie *Monitor* aus. Da wir auch die Threads betrachten wollen setzen Sie nun einen Hacken bei *Enable threads monitoring.*
4. Mit einem Klick auf *Run* startet der Profiler auch schon. An der Stelle wo Sie eben noch durch Ihr Projekt navigieren konnten ist nun ein neuer Reiter erschienen. Wählen Sie diesen neuen Punkt *Profiler* aus. Unter dem Punkt *View* finden Sie eine Graphik mit dem Titel *Threads*, wenn Sie diese ausgewählt haben erscheint auf der rechten Seite eine Übersicht über die aktuell existierenden Threads und deren Status.
5. Mit dem Starten des Profilers wurde ebenso das Programm gestartet. Klicken Sie in dem Programm *SyncFiles* nun einmal auf *Analyse*. Sie werden feststellen, dass ein neuer Thread in unserer Übersicht erschienen ist. Klicken Sie nun auf *Abbruch.* Der neue Thread, der eben noch lief ist nun wieder gestoppt worden.
6. Ein einzelner Thread kann also gestartet und wieder gestoppt werden. Wie sieht es nun mit mehreren Threads aus? Probieren wir dies nun aus. Dazu klicken Sie mehrmals auf den Punkt *Analyse* in dem Programm *SyncFiles*. Es erscheinen, wie erwartet, mehrere neue Threads in der Übersicht. Klicken Sie nun auf *Abbruch*, Sie werden feststellen, dass nur der neueste Thread abgebrochen wurde und die älteren Threads weiter laufen. Wie könnte dies nun also verhindert werden?
7. Beenden Sie nun den Profiler, indem Sie entweder das Programm einfach schließen, oder indem Sie unter dem Menüpunkt *Profiler* auf *Stop Profiling Session* klicken.

8. Wählen Sie die Datei *SyncFilesView* im Design-Modus auf. Nun wählen Sie den Button *Analyse* aus. Den Eigenschaften können wir entnehmen, dass der Button *jButtonAnalyse* heißt. Durch einen Doppelklick auf den Button gelangen wir zu dem Quellcode, welcher ausgeführt wird, wenn der Button gedrückt wird.
9. Durch deaktivieren des Buttons ist es dem User nicht möglich einen weiteren Thread zu starten. Daher fügen wir nun *jButtonAnalyse.setEnabled(false);* beim starten der Analyse ein.
10. Da der Button auch wieder aktiviert werden muss führen Sie bitte den Punkt 8 und Punkt 9 analog mit dem *Abbruch* Button durch. Nun wird bei der Funktion *setEnable* ein *true* gesetzt.

9.5. Quellen

Informationen:
http://profiler.netbeans.org/
http://netbeans.org/download/flash/netbeans_60/profiler/profiler.html
http://netbeans.org/kb/docs/java/profiler-profilingpoints.html
http://netbeans.org/kb/docs/java/profiler-intro.html
http://www.drdobbs.com/java/184406433
http://netbeans.org/download/flash/netbeans_60/profiler/profiler.html
http://wiki.netbeans.org/NetBeansUserFAQ#section-NetBeansUserFAQ-Profiler

Bilder:
In Netbeans erstellt.
Teilweise aus dem Programm „SyncFiles“ von:http://home.arcor.de/ka.menzel/

10. CVS / Local History - Christian Kropp, 29.11.2010

10.1. Was ist eigentlich CVS?

CVS steht für „Concurrent Version System“ und ist ein System zur Versionsverwaltung / Versionskontrolle. Es wurde für Softwareprojekte entwickelt. Als Versionierungs-Tool soll es bei folgenden Problemen als Werkzeug dienen:

- mehrere Benutzer arbeiten gleichzeitig an einem Projekt und bearbeiten die gleiche Datei, dabei tritt ein Konflikt auf
- in einem Programm tritt ein Bug auf — seit wann gibt es den Bug, wie ist er entstanden?
- Entwickler möchten verteilt über das Internet ein Programm entwickeln

Generell lässt sich also das folgende Anforderungsprofil für ein Versionsmanagementsystem definieren:

- Änderungen in Quelltexten sollen ersichtlich gemacht werden
- alle Änderungen in Quelltexten sollen dokumentiert werden
- alle älteren Versionen sollten wiederherstellbar sein
- mehrere Benutzer sollen gleichzeitig an einem Projekt arbeiten können

Daraus kann man schon mehrere Dinge erkennen:

1. Das CVS ist ein Multiuser-Tool, der es ermöglicht ein Projekt auf mehreren Computern / Schnittstellen zu benutzen,
2. Ältere Versionen von einer Datei werden zur Nachkontrolle gespeichert,
3. Änderungen werden an einer zentralen Stelle für jeden ersichtlich gespeichert, damit die verschiedenen Versionen nachvollzogen werden können.

Doch das ist noch längst nicht alles, denn das CVS bietet als Freeware-Tool durch seine Client-Server-Struktur einen zentralen Anlaufspunkt, das Repository (engl. Behälter). In diesem werden die verschiedenen Projekte einer Firma gespeichert und können von den einzelnen Anwendern (den Clients) heruntergeladen werden. Optional können die einzelnen Projekte durch Passwörter geschützt werden, womit Fremdzugriff z.B. einer anderen Abteilung vermieden werden kann.
Die Client-Server-Anwendung kann man sich in den folgenden Abbildungen näher ansehen:

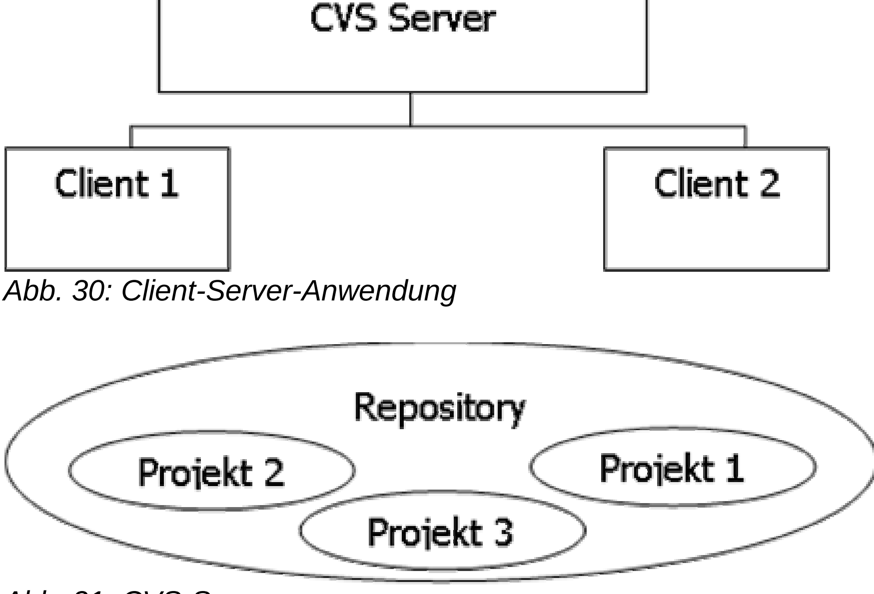


Abb. 30: Client-Server-Anwendung

Abb. 31: CVS-Server

Wie in Abbildung 1 zu erkennen ist, handelt es sich beim CVS-Server um einen zentralen, von jedem Client zugänglichen Server. Auf diesen kann jeder (mit Einschränkung des Passwortschutzes) zugreifen und sich seine Projekte lokal speichern wie in Abbildung 2 zu sehen ist.
Das Repository legt auf dem Server pro Projekt ein Verzeichnis ab, indem alle Dateien und die allgemeinen Konfigurationsparameter hinterlegt sind. Zusätzlich werden noch die folgenden Informationen für das Projekt bzw. die einzelnen Dateien des Projektes festgelegt:

- Unterschiede zu den vorherigen Versionen
- Änderungszeitpunkt
- Kommentare und Benutzername des Clients
- Versionsnummer

Im Folgenden wird der Zugriff auf das CVS-System erläutert.
Beginnend dazu ist ein Beispiel aufgeführt, die den typischen Ablauf eines Imports mitsamt einem Checkout von einem Client mit seinem Repository aufführt. Die einzelnen Begrifflichkeiten werden im Anschluss näher erläutert.

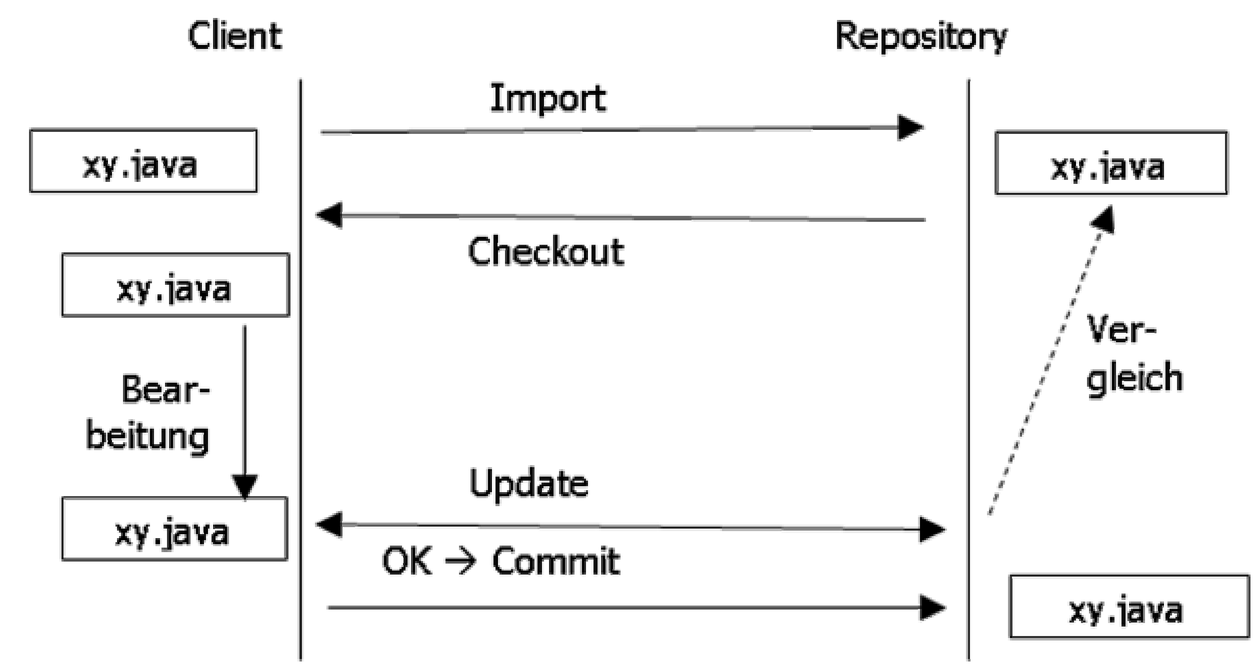


Abb. 32: Import und Checkout zwischen einem Client und dem Repository

Das Beispiel führt das Importieren, das Hochladen einer Datei (in diesem Fall) vom Client zum Repository, und das anschliessende Checkout, das Herunterladen einer Datei vom Repository zum Client, aus.
Im Anschluss wird die Datei lokal vom Client bearbeitet und Änderungen unterzogen. Zum Beispiel hat der Benutzer eine Funktion vergessen oder führt entsprechende Ergänzungen zu einer bereits bestehenden Funktion hinzu.
Wenn er dann die Datei wieder auf das Repository hochlädt, stellt selbiges mittels dem CVS fest, dass es bei den Versionen sowie im Quellcode selbst Unterschiede gibt. Wird die Ausgangsdatei als die selbe ausgemacht und gibt es bis auf die Änderungen des Benutzers keine anderen Ergänzungen, Löschungen oder sonstiges, dann aktualisiert das Repository ihre vorhandene Version mit der des Clients und protokolliert die bestehende Version als Vorgängerversion. Gleichzeitig damit wird die Versionsnummer hochgesetzt. Optional kann beim Commit des Clients eine Nachricht hinterlegt werden, welche Änderungen vorgenommen worden sind. Dieser Text wird mit der Datei auf dem CVS-Server gespeichert und kann mitsamt den vorherigen Versionen von anderen Benutzern angesehen werden.

Um jetzt nicht noch weiter Fachbegriffe zu verwenden, die im CVS vermehrt auftauchen, ohne eine komplette Erklärung zu liefern, werden diese in der Folge näher erläutert. Hierzu

sei gesagt, dass es nur eine Auswahl der gängigsten Befehle ist. Das CVS umfasst annähernd 50 Befehle, wovon die meisten noch auf die textuelle Verarbeitung des CVS-Systems zurückgehen.

- Checkout: Eine Arbeitskopie vom Repository holen
- Import: Ein neues Projekt anlegen / Verzeichnis in das Repository übergeben
- Add: Neue Dateien in das Projekt / Verzeichnis einstellen
- Update: Die eigene Arbeitskopie aktualisieren, indem die Änderungen anderer Nutzer abgerufen werden
- Commit: Eigene Änderungen zum Repository senden
- Log: Kommentare, die beim Check-In durch den Anwender verfasst werden und die vorgenommenen Änderungen beschreiben
- Diff: Anzeigen der Unterschiede zweier Dateiversionen

Nachdem nun die Begrifflichkeiten geklärt sind, wollen wir einen Blick darauf werfen, was passiert, wenn zwei Clients ein und die selbe Datei vom Repository herunterladen und lokal bearbeiten.

Auf der folgenden Seite wird das obige Beispiel (Abbildung 3) durch einen zweiten Client ergänzt, der ebenfalls die Datei vom Repository herunterlädt. Beide Clients bearbeiten ihre Datei lokal auf ihrem PC und soweit der erste Client seine lokale Kopie auf dem Repository hochladen will, ist noch alles in Ordnung.
Sobald aber der zweite Client im Anschluss seine Version mit der Version auf dem Repository abgleicht, stellt er Unterschiede in der Versionierung sowie beim Quellcode fest. Änderungen, die von Client 1 gemacht wurden, waren in der Ausgangsdatei von Client 2 noch nicht vorhanden und führen somit zu einem Konflikt, der manuell vom Benutzer behoben werden muss. Was hieran auffällt, ist, dass der Vergleich beim zweiten Update von Client 2 mit der aktualisierten Version von Client 1 erfolgt.

Die Unterschiede zweier Versionen können mittels des Diff-Befehls festgestellt werden und in den verschiedenen Applets gibt es Tools, die solche Konflikte lösen. Zu einem späteren Zeitpunkt wird dabei im Abschnitt „CVS in Netbeans“ näher drauf eingegangen.

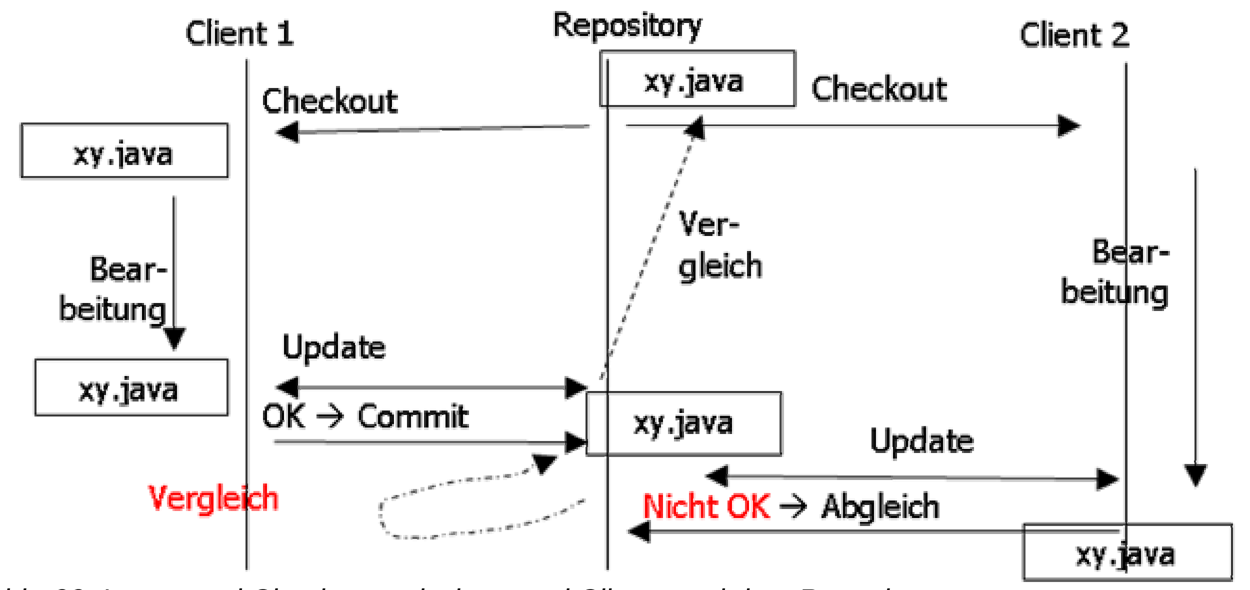


Abb. 33: Import und Checkout zwischen zwei Clients und dem Repository

10.2. Das ist CVS nicht!

Im ersten Abschnitt haben wir uns damit beschäftigt, was CVS eigentlich ist und alles kann. Nun kommen wir dazu, was CVS nicht kann und wozu es nicht dient oder wozu die Funktionen schlichtweg nicht ausreichen.
Denn zum Einen erleichtert es die Teamarbeit an einem Projekt, doch zum Anderen gibt es bestimmte Dinge, die immer noch erledigt werden müssen. Dazu gehören unter Anderem:

- CVS ist kein Erstellungssystem, d.h. das CVS legt nicht fest, unter welchem Pfad oder wo ein Projekt gespeichert wird
- Es gibt darüber hinaus keine Vorschriften zur Speicherung
- Es wird auch nicht festgelegt, wie der vorhandene Festplattenspeicher am besten ausgenutzt werden kann
- Es bietet keine Hilfe im Allgemeinen
- Es hat keine Änderungskontrolle oder hinterfragt Änderungen, die von Clients auf dem Repository hinterlegt werden
- Im Allgemeinen ist das CVS kein Management- oder Kommunikations-Ersatz; es können keine Team-Meetings durch das CVS ersetzt werden oder Fragen durch das CVS beantwortet werden
- Die gemeinsamen Änderungen von einem Client können nicht nachvollzogen werden. Das Repository kann immer nur die einzelnen Änderungen an einer Datei protokollieren; welche Dateien zusätzlich dazu noch beim Commit geändert oder hinzugefügt worden sind, ist dem CVS nicht bekannt
- CVS ist kein automatisches Test-Programm im logischen oder programmiertechnischen Sinn. Es können auch Fehler im Quellcode auf dem Repository hochgeladen werden, was zu keinen Beeinträchtigungen im Projekt führt – bis auf die Tatsache, dass es möglicherweise nicht funktioniert

Es sollte gesagt werden, dass manche aufgeführten Punkte einleuchtend sein sollten. Erweiterungen und Erläuterungen dazu bietet auch der Abschnitt „Vor- / Nachteile von CVS“.

10.3. Geschichte des CVS

Die eigentliche Historie des CVS beginnt im Jahre 1986, als Dick Grune erste Code-Segmente veröffentlichte.
Diese, als Grundlage diente das bisherige Verwaltungssystem bestehend aus RCS (Revision Control System) und SCCS (Source Code Control System), sollte den zu diesem Zeitpunkt Status Quo ablösen. Das Projekt CVS wurde in den folgenden Jahren konsequent weiterentwickelt und im Jahr 1989 entstand CVS endgültig.

Die Version 1.0 wurde am 19. November 1990 finalisiert.

CVS nutzt bei seiner Versionierung das gleiche Datenformat wie RCS und ist insgesamt sehr an seinen Vorgänger angelehnt.
Die ersten Versionen waren dabei noch auf Kommandozeilen-Basis und erst in späteren Versionen wurde eine graphische Oberfläche der Standard des CVS.
Heutzutage wird CVS vorrangig bei der Entwicklung von Open-Source-Projekten eingesetzt, wobei auch dort neuere Versionierungs-Software zum Einsatz kommt, womit sich im Abschnitt „Alternativen zu CVS“ beschäftigt wird.

Zum CVS sei noch gesagt, dass die Entwicklung seit ca. 2006 bzw. 2008 nicht mehr in seinem vollem Umfang betrieben wird. Es werden zwar noch einzelne Bugfixes vorgenommen, aber an einem kompletten Versionsupdate wird nicht mehr gearbeitet.

10.4. CVS in Netbeans

Wenn in den bisherigen Abschnitten Fragen an der Vorgehensweise des CVS aufgetreten sind, dann werden sie in diesem Abschnitt hoffentlich beantwortet werden können.

Das CVS soll hier in seiner ganzen Funktionsweise erläutert werden und die einzelnen Schritte vom Checkout vom Repository, Commit, Konfliktlösung bei verschiedenen Versionen und Diff-Anzeige wird einzeln erläutert.

Doch erstmal ein wenig zur Geschichte von CVS in Netbeans:
CVS war das erste Versionierungs-Tool, das von Netbeans standardmäßig unterstützt wurde. In der heutigen Version von Netbeans werden zusätzlich noch die Versionierungs-Tools Subversion, Mercurial und Kenai unterstützt (dazu später mehr im Abschnitt „Alternativen zu CVS“).

Die älteren Versionen von Netbeans werden im CVS abgespeichert, während der Sourcecode mit Mercurial gesichert wird.

Soviel zur geschichtlichen Entwicklung.
Jetzt wird die komplette Abwicklung eines Projektes vom Checkout über den Import bis zum Commit und den sonstigen Funktionen von CVS am Beispiel von Netbeans besprochen.

Das Ganze startet mit dem Checkout vom Repository, wobei erstmal

- Der Reiter „Team“ und dort der Eintrag „CVS“ gewählt werden muss
- Anschliessend wird „Checkout“ angeklickt
- Jetzt wird der Servername mit einem Pfad bestimmt; evtl. ist hier ein Passwortschutz eingebaut
- Danach kann man einen Ordner auswählen

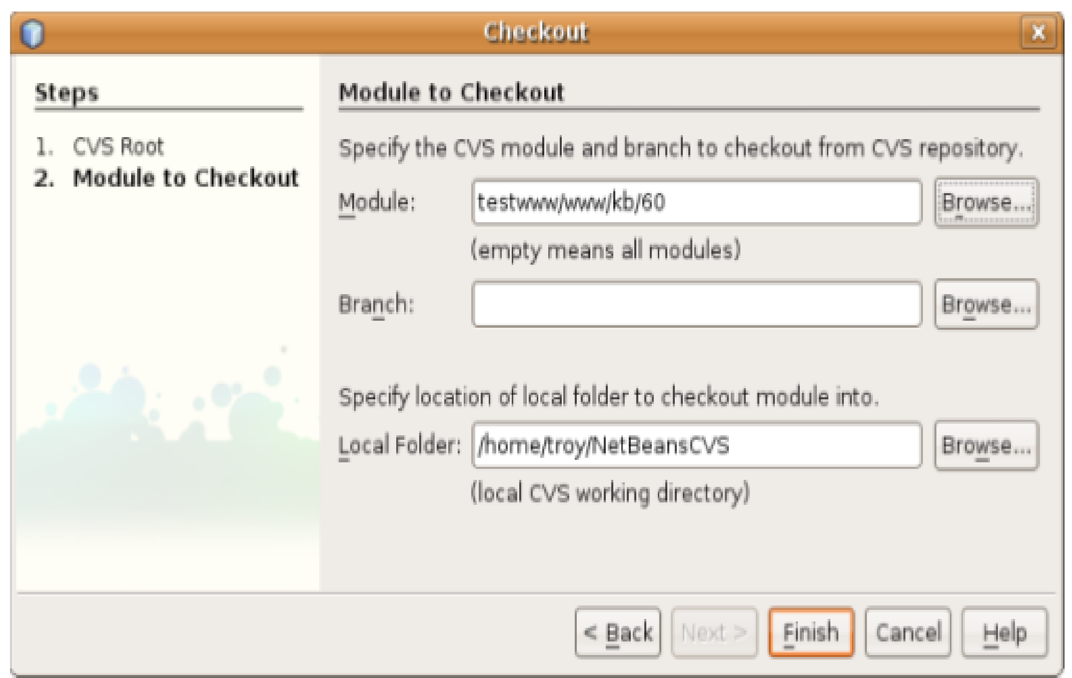


Abb. 34: Checkout

Kommen wir im Anschluss an die sogenannte Farblehre bei Projekten, Paketen, Ordnern und Dateien.
Bei Projekten, Paketen und Ordnern gibt es zwei Unterscheidungen:

1. Blau: Bei Änderungen
2. Rot: Bei Konflikten

Bei Dateien kann es zu mehr unterschiedlichen Farbwerten kommen:

- Blau: Änderung
- Grün: Lokal hinzugefügt
- Rot: Konflikt
- Grau: Ignoriert durch CVS (z.B. bei Löschung)
- Durchgestrichen: Ausgenommen von Commit-Operationen

Als nächster Schritt erfolgt das Importieren aufs Repository, welche folgende Grafik am besten veranschaulicht.

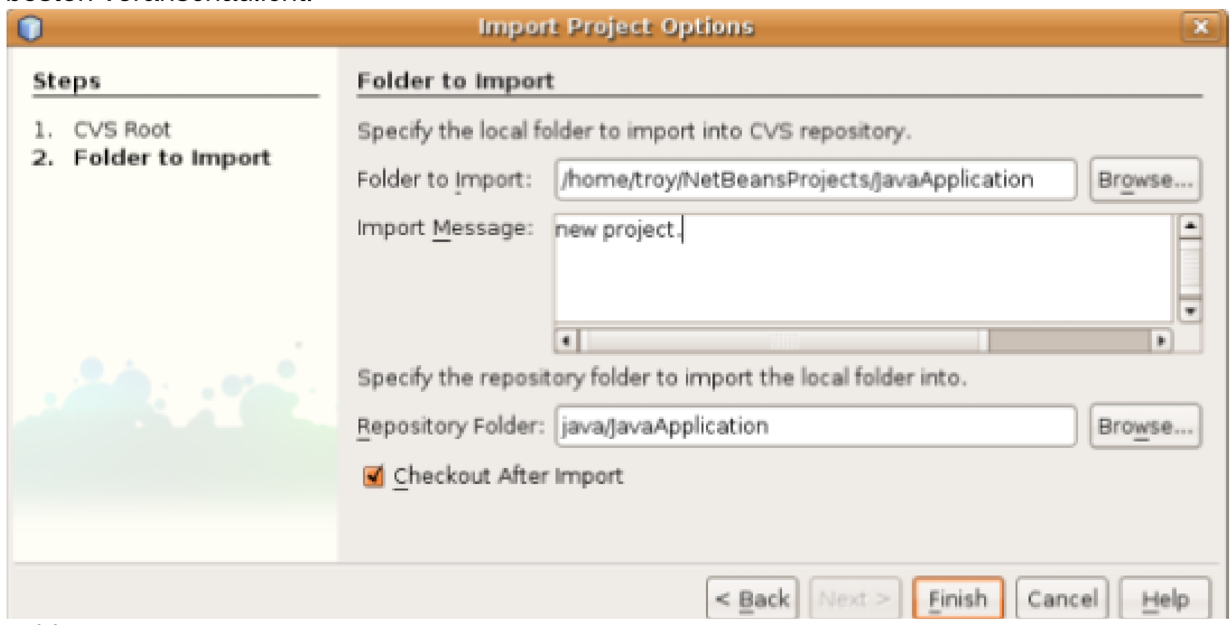


Abb. 35: Import

Als zusätzliche Hilfe dient beim CVS eine Auflistung der Dateien. Diese wird unter dem Quellcode-Editor angezeigt und sieht folgendermaßen aus:

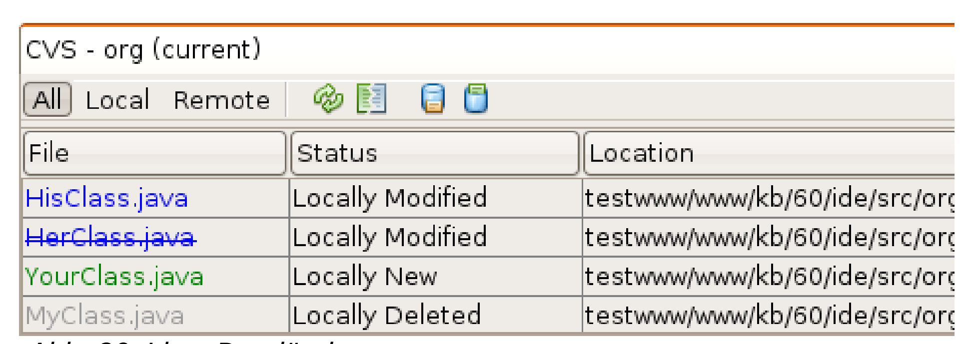


Abb. 36: Liste Dateiänderungen

Die einzelnen Symbole bedeuten (von links nach rechts):

- Aktualisierung aller aufgelisteter Dateien
- Gegenüberstellung lokale Version <--> Version auf Repository
- Update aller lokalen Dateien vom Repository ausgehend
- Commit aller lokalen Dateien auf das Repository

Gerade angesprochen ist auf folgender Abbildung zu sehen, wie ein Commit-Fenster aussieht:

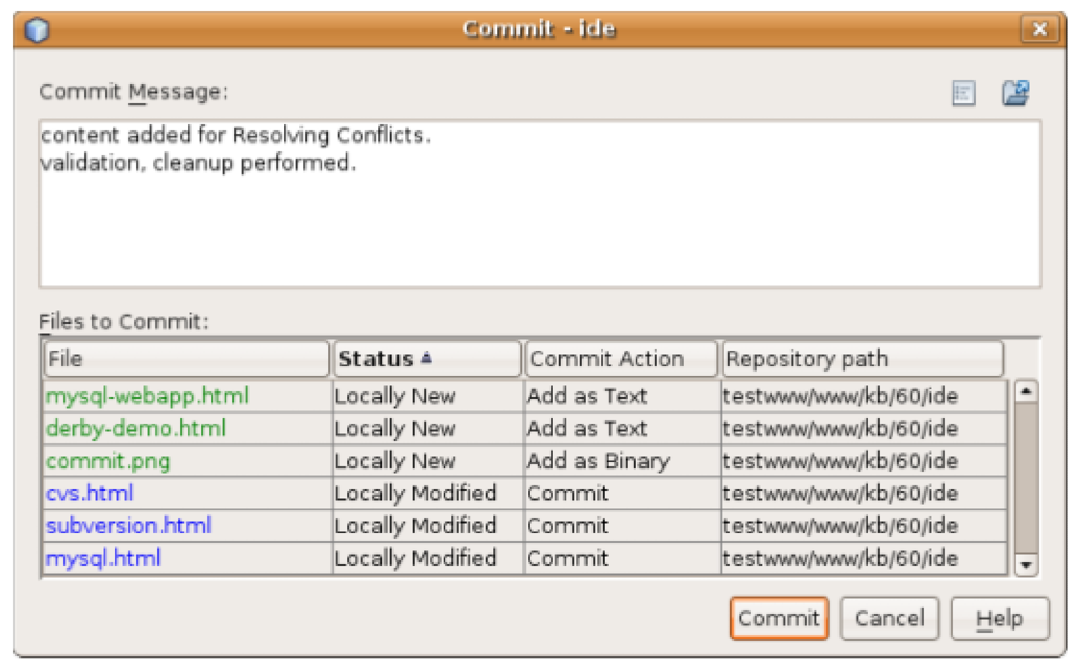


Abb. 37: Commit

In den letzten beiden Abbildungen wird zum Einen gezeigt, wie das Diff-Vergleichsfenster die lokale Version und die Version auf dem Repository gegenüberstellt, und zum Anderen, wie bei einem Konflikt vom CVS vorgegangen wird. Der Abgleich bei einem Konflikt muss dabei manuell erfolgen.

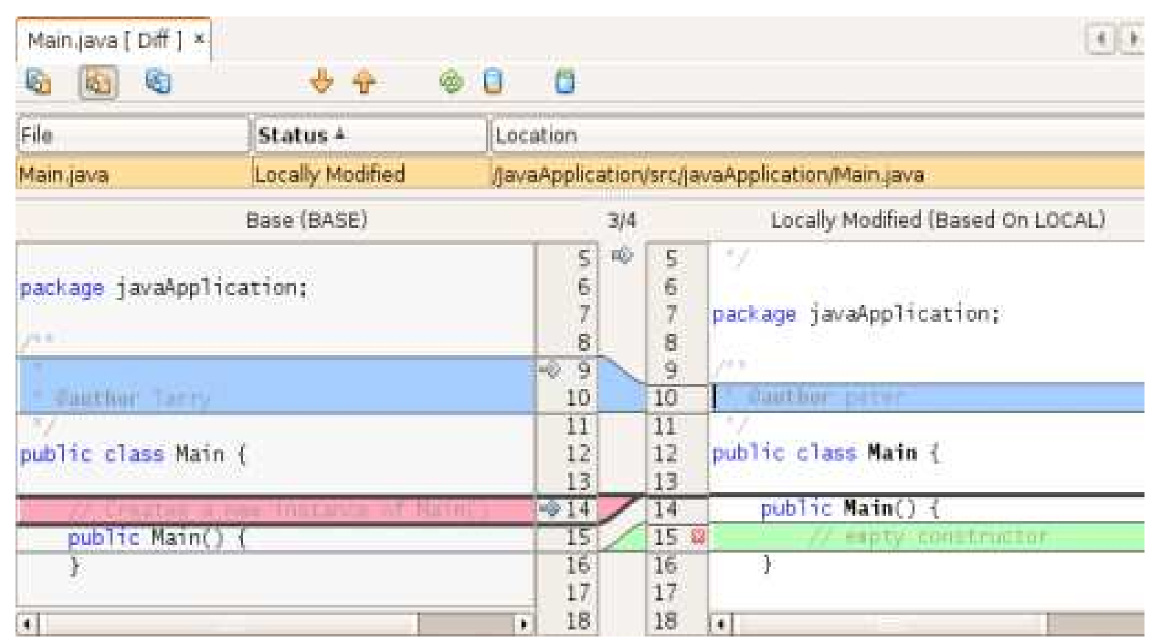


Abb. 38: Diff-Anzeige

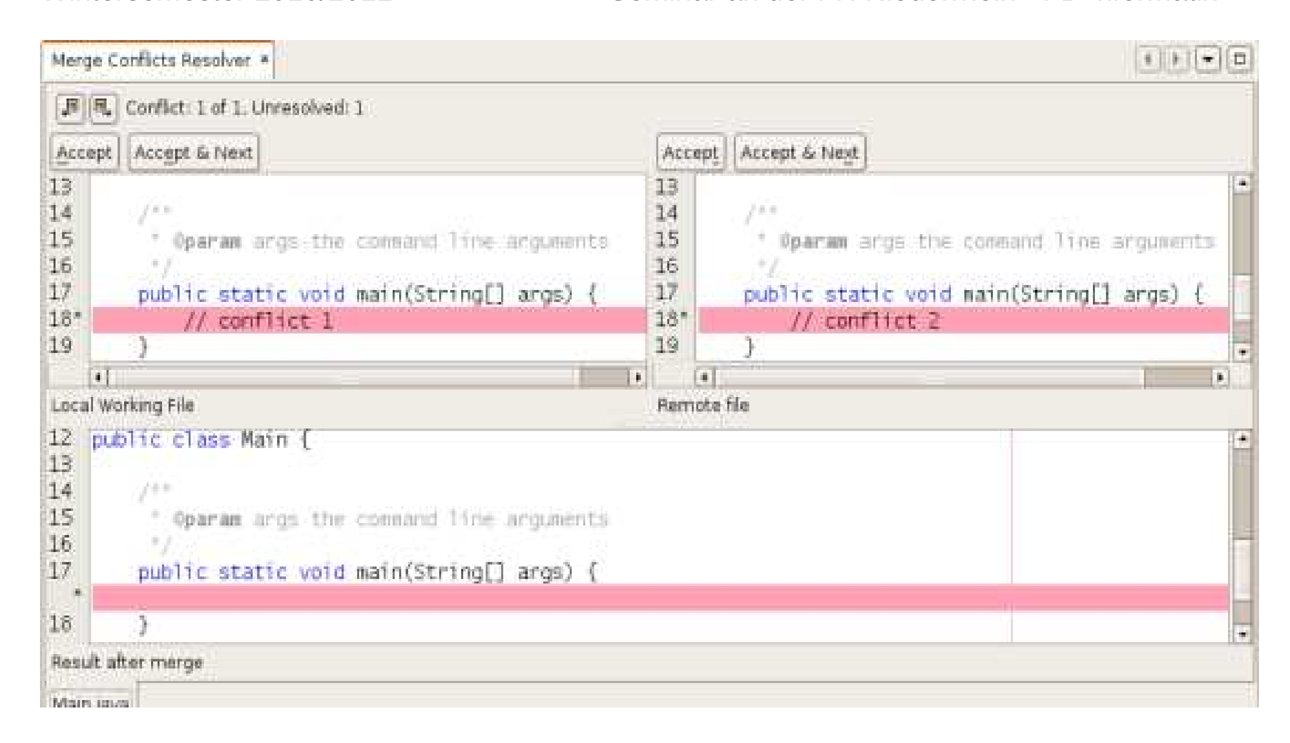


Abb. 39: Merge Conflicts Resolver

10.5. Vor- / Nachteile von CVS

In den vorherigen Abschnitten wurde die Funktionsweise von CVS beschrieben und auch ein Einblick darüber, was CVS ist und was es nicht ist, wurde gegeben. Hier werden nun einige Vor- und Nachteile aufgeführt, die dieses Versionierungs-Tool bietet oder woran es einen Nachteil zu anderen Versionierungs-Tools aufweist.

Vorteile:

- Gleichzeitiges Bearbeiten von Dateien
- Automatische Abgleichung von Änderungen und Konflikten
- Versionierung
- (relativ) sicher, d.h. das Repository kann durch Sicherheitsmassnahmen (Passwortschutz) vor Fremdzugriffen geschützt werden
- Logging
- Wiederherstellung älterer Dateistände
- Open Source / Freeware

Nachteile:

- Umbenennen / Verschieben wird nicht berücksichtigt
- Keine Versionierung von symbolischen Links
- Textuelle Abspeicherung der Dateien; damit treten bei Binärdateien beim Abspeichern Probleme auf

10.6. Alternativen zu CVS

Über Vor- und Nachteile wurde im vorangegangenen Abschnitt gesprochen. Nun kommen wir zu den Alternativen in den Versionierungs-Tools, die es gibt. Dabei haben die beiden Erstgenannten (Subversion und Mercurial) das CVS als Standard zum größten Teil abgelöst. Nun werden die einzelnen Alternativen kurz vorgestellt und erläutert:

10.6.1. (Apache) Subversion

Die Versionierung erfolgt in einem zentralen Projektarchiv (engl. repository) in Form einer einfachen Revisionszählung. Wenn Änderungen an Inhalten verteilt auf den Computern der Bearbeiter ausgeführt werden, werden zwischen dem Projektarchiv und einem Arbeitsplatz jeweils nur die Unterschiede zu bereits vorhandenen Ständen übertragen.
Subversion wird als Freie Software unter einer Lizenz im Stil der Apache-Lizenz veröffentlicht. Die Benennung „Subversion“ setzt sich aus den Worten Sub und Version zusammen und kann so als Unterversion, frühere Version verstanden werden. Eine andere Interpretation führt den Namen auf den politisch-soziologischen Begriff Subversion (etwa: Umsturz) zurück. Allerdings ist von den Entwicklern nicht dokumentiert worden, ob dieses Wortspiel beabsichtigt ist.
Subversion versteht sich als Weiterentwicklung von CVS und entstand als Reaktion auf weit verbreitete Kritik an CVS. In der Bedienung der Kommandozeilenversion ist es sehr ähnlich gehalten. Mit Subversion ist es aber – im Gegensatz zu CVS – z. B. möglich, Dateien oder Verzeichnisse zu verschieben oder umzubenennen, ohne die Versionsgeschichte zu verlieren.

10.6.2. Mercurial

Es wird nahezu vollständig in Python entwickelt: lediglich eine diff-Implementierung, die mit binären Dateien umgehen kann, ist in C umgesetzt. Mercurial wird primär über die Kommandozeile benutzt; alle Kommandos beginnen mit „hg“, dem Elementsymbol von Quecksilber (engl. Mercury). Entwicklungsschwerpunkte von Mercurial sind Effizienz, Skalierbarkeit und robuste Handhabung von Text- und Binärdateien. Während bei Versionskontrollsystemen mit zentralem Server (wie CVS oder SVN) Dritte in der Regel nur lesenden Zugriff auf das Repository haben, wird bei Mercurial das Repository des Projektes, an dem man entwickeln will, „geklont“, also eine lokale Kopie erstellt. Auf dieser lokalen Kopie stehen dann die üblichen Funktionen zur Verfügung, beispielsweise das Erstellen neuer Revisionen, Change Set genannt.

Die Fähigkeit, Entwicklungszweige zu erstellen und zusammenzuführen (engl.: „branching“ und „merging“), ist fester Bestandteil von Mercurial. Eine integrierte Web-Schnittstelle steht zur Verfügung; Drittanbieter stellen grafische Frontends oder Plugins für Entwicklungsumgebungen zur Verfügung.

10.6.3. Kenai

Nutzt CVS-, Subversion- & Mercurial-Repositories

10.6.4. CVSNT

Ist eine Weiterentwicklung zu CVS. Alle Änderungen werden in einem Set von Datei gesichert – typischerweise in der Implementation eines Software-Projekts. Es ist kompatibel mit dem CVS.

10.6.5. EVS

EVS ist eine Client-Server-Anwendung, die Änderungen in Dateien als Teil eines Konfigurations-Management-Prozesses speichert. Der Server-Teil kann auch in Standalone-Manier auf dem Client betrieben werden.
EVS ist ein erweitertes Mulitplattform-Versionskontroll-System. Es richtet sich an den bestehenden Standard des CVS-Protokolls und hat neben der Robustheit viele zusätzliche Features für moderne Betriebssyteme und Clients.

10.6.6. OpenCVS

Es kann als Client und als Server für Repository eingesetzt werden und bietet detaillierte Zugriffskontrollen für Daten, die in dem Repository abgelegt worden sind. Es versucht, so kompatibel wie möglich zu anderen CVS-Implementationen zu sein – es sei denn, bestimmte Funktionalitäten würden die Gesamtsicherheit des Systems beeinträchtigen.

Das OpenCVS-Projekt wurde nach den Diskussionen über die zuletzt bekannt gewordenen GNU-CVS-Sicherheitslücken begonnen. Obwohl CVS weitverbreitet ist, stand die Entwicklung für die letzten Jahre beinahe still und viele Sicherheitsprobleme traten auf; sowohl in der Implementation als auch im Mechanismus. Bislang ist OpenCVS noch nicht veröffentlicht.

10.7. Local History (in Netbeans)

Im letzten Abschnitt geht es um die Local History, die im Applet Netbeans seit der Version 6.0M7 fester Bestandteil ist. Die Local History ist sozusagen die lokale Versionierung der

Dateien, indem vorangegangene Änderungen lokal abgespeichert und wiederhergestellt bzw. verglichen werden können.
Um die Local History aufzurufen, kann man mittels Rechtsklick auf die entsprechende Datei unter dem Punkt „Local History“ die hinterlegten Speicherstände begutachten.

Abb. 40: Local History

Im Optionsmenü können die Einstellungen zur Local History vorgenommen werden und festgelegt werden, wie lange die Dateien gespeichert werden und wie viele Versionen zur Verfügung gestellt werden, was man folgender Abbildung entnehmen kann.



Abb. 41: Optionen Local History

In der Versionsanzeige kann man dann ältere Versionen im Vergleich zu seiner aktuellen Version sich anzeigen lassen und die Änderungen ganz simpel mit einem Klick wieder rückgängig machen.

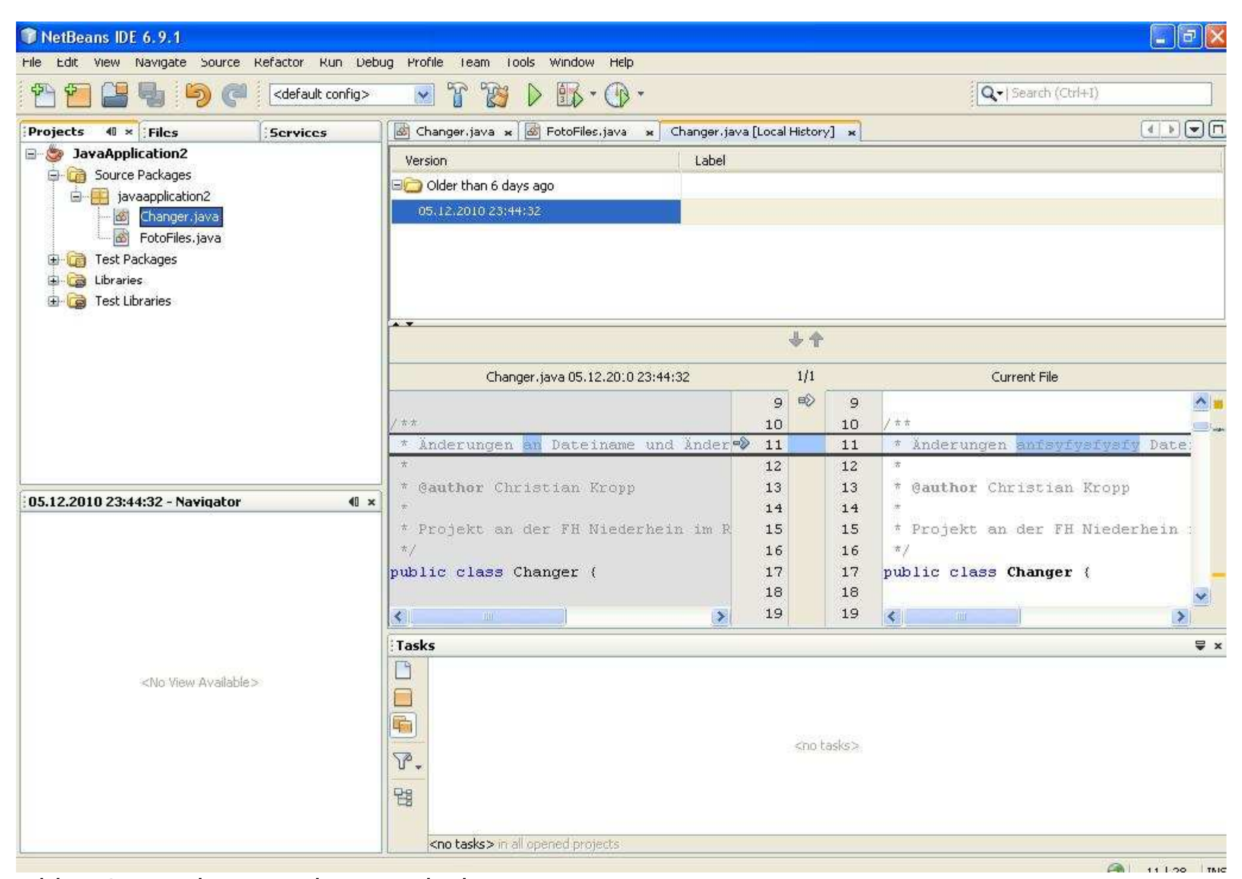


Abb. 42: Versionsanzeige Local History

11. Javadoc - Alexander Nolting, 29.11.2010

11.1. Software-Dokumentation

11.1.1. Was ist eine Dokumentation?

In Zusammenhang mit Software gibt es zwei verschiedene Arten von Dokumentation.
Zum einen die Beschreibung der Benutzerschnittstelle eines Programms, also eine Beschreibung, wie welche Funktion eines Programmes benutzt wird (sowie Hilfestellungen bei Fehlfunktionen). Eine solche Dokumentation liegt in Form eines Handbuches oder der Hilfe-Funktion (F1) vor. Diese Art der Dokumentation richtet sich an den *Anwender* eines Programms.
Zum anderen gibt es die Beschreibung des Quellcodes eines Programms. Diese Art der Dokumentation richtet sich an den *Programmierer* und soll von nun an von uns von Bedeutung sein.

11.1.2. Dokumentation des Quellcodes

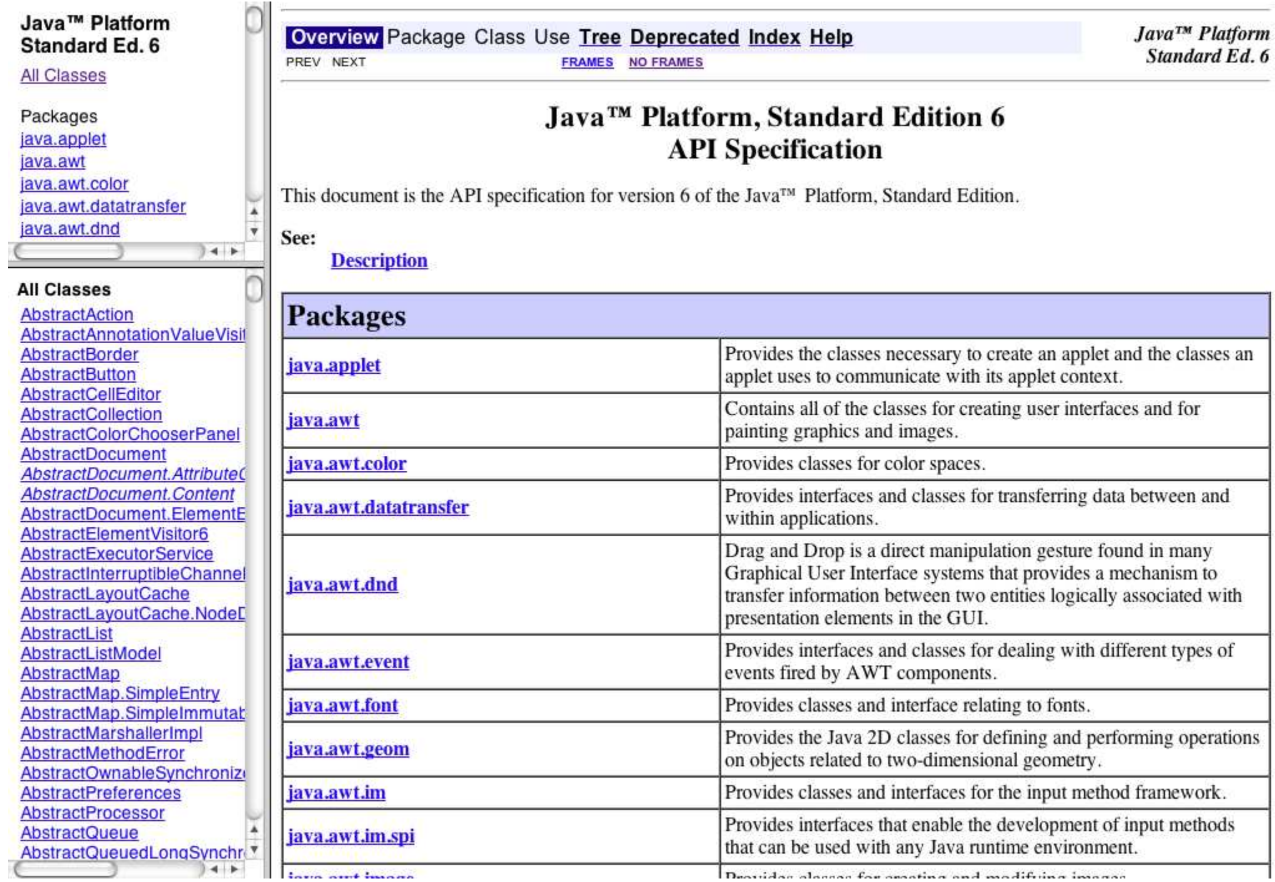
Eine Dokumentation des Quellcodes beinhaltet die Beschreibung der verwendeten Variablen, Methoden (hier vor allem die Schnittstelle, also die public-Methoden, da dies auf Quellcode-Ebene die Funktionalität ist, die „von außen“ zur Verfügung steht), Klassen, Exceptions usw.
Ferner kann diese Dokumentation weitere Informationen enthalten, wie zum Beispiel Angaben über den Autor, eine (kurze) Beschreibung des Programms bzw. der Programmteile und Abschätzungen über das Laufzeitverhalten.
Welche Angaben eine Dokumentation enthalten sollte, hängt davon ab, was (für das gesamte Projekt) sinnvoll erscheint, hier gibt es keine festgelegte Norm. Sie sollte allerdings so kurz wie möglich gehalten werden.

11.1.3. Wozu dient eine Dokumentation?

Heutzutage sind meist mehrere Personen an einem Software-Projekt beteiligt. Dies erfordert, dass Programmteile von anderen Programmierern verstanden werden müssen. Nun ist Quellcode (vor allem, der nicht von einem selbst geschrieben wurde) sehr schwer zu verstehen. Und da die beteiligten Programmierer jedoch räumlich und zeitlich stark von einander getrennt sein können (dh. ein Mitarbeiter kann in einem anderen Land sitzen, oder ein Teil des Quellcodes kann schon vor einem Jahr geschrieben worden sein), werden persönliche Absprachen und Erklärungen schwierig. Somit ist zusätzliche Dokumentation erforderlich.
Die Dokumentation erleichtert eine Wiederverwendung von Klassen und Paketen, da anhand der Dokumentation schneller ausgemacht werden kann, ob eine benötigte Funktionalität schon zur Verfügung steht, oder in einem anderen Projekt bereits enthalten ist. Auch wird das Prinzip der Datenkapselung unterstützt, da eine Dokumentation die Schnittstelle spezifiziert. So hat man einen „Leitfaden“, dem bei der Implementation gefolgt werden muss. Somit werden auch nachträgliche Änderungen erleichtert.
Dokumentation ist außerdem eine wichtige Programmierrichtlinie, deren Einhaltung die Qualität einer Software erhöht.

11.1.4. Beispiele

- *Die OpenCV-Dokumentation*
  http://opencv.willowgarage.com/documentation/index.html
- *Die Dokumentation der Java-API*
  http://download.oracle.com/javase/6/docs/api/index.html



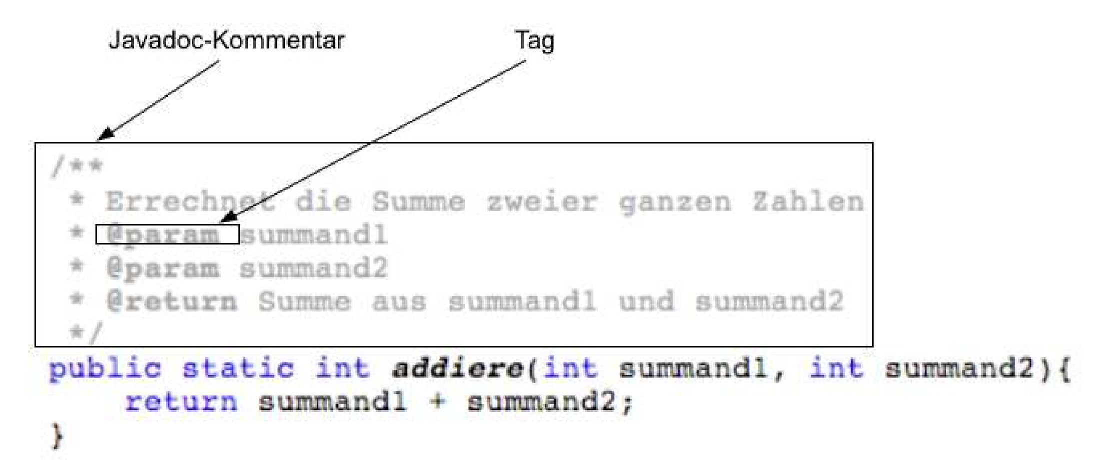
Diese Dokumentation wurde mit Javadoc erstellt.

11.2. Was ist Javadoc?

Javadoc ist ein Hilfsprogramm, das (aus dem Quellcode heraus) automatisch eine Dokumentation erstellt. Da dies automatisch geschieht, stellt Javadoc eine Arbeitserleichterung gegenüber separater, manueller Dokumentation dar.
Javadoc wurde genau wie Java von Sun (bzw. Oracle) entwickelt und ist seit Version 2 im JDK von Sun/Oracle enthalten. Es wird mittlerweile als Industriestandard angesehen und ist deshalb auch in den meisten IDE's enthalten, so auch in Netbeans.

11.3. Funktionsweise von Javadoc

Javadoc erstellt die Dokumentation aus dem Quellcode heraus. Dazu wird vorher (während des Programmierens) der Quellcode mit bestimmten Kommentaren (so genannten „Tags") angereichert, die dazu dienen, Elemente im Quellcode (Variablen, Methoden, Klassen usw.) naher zu beschreiben. Javadoc parst diese Tags und erstellt daraus eine Dokumentation. Diese Dokumentation wird standardmäßig im HTML-Format erstellt.



Javadoc-Kommentare müssen mit /** beginnen und unmittelbar vor dem betreffenden Element eingefügt werden. Tags beginnen mit @ (oder mit {@).
Folgend ein Auszug der existierenden Tags.

11.3.1. Tags (Auszug)

`@author -Name-`	Gibt Autor des Programms an
`@version -Nr-`	Gibt Version eines Programms oder Klasse an
`@see -Element-`	Erzeugt Link auf anderes Element
`@param -Name- -Beschreibung-`	Parameterbeschreibung Methode
`@return -Beschreibung-`	Beschreibung Rückgabewert einer Methode
`@exception -Exception- -Beschreibung-`	Beschreibung einer Exception, die von einer Methode geworfen werden kann
`@deprecated`	Deklariert Methode als veraltet → sollte nicht mehr benutzt werden
`{@literal}`	Unterdrückt Interpretierung von Tags (innerhalb der Klammern)
`{@code}`	Wie {@literal}, nur dass Quellcodezeichensatz verwendet wird
`{@inheritDoc}`	Kopiert Beschreibung der überschriebenen Methode

11.3.2. Dokumentation erstellen

Wird eine Dokumentation mit Javadoc erstellt, werden mehrere Dateien erzeugt (siehe 3.3). Diese werden alle in einem bestimmten Verzeichnis erzeugt. Javadoc kann mit bestimmten Optionen angepasst werden, sodass man zum Beispiel selbst definierte Verzeichnisse angeben kann, in die die Dokumentation erstellt werden soll.
Es gibt zwei Wege, die Dokumentation zu erstellen. Zum einen in der Konsole, zum anderen in einer IDE (Netbeans)

- *In der Konsole*

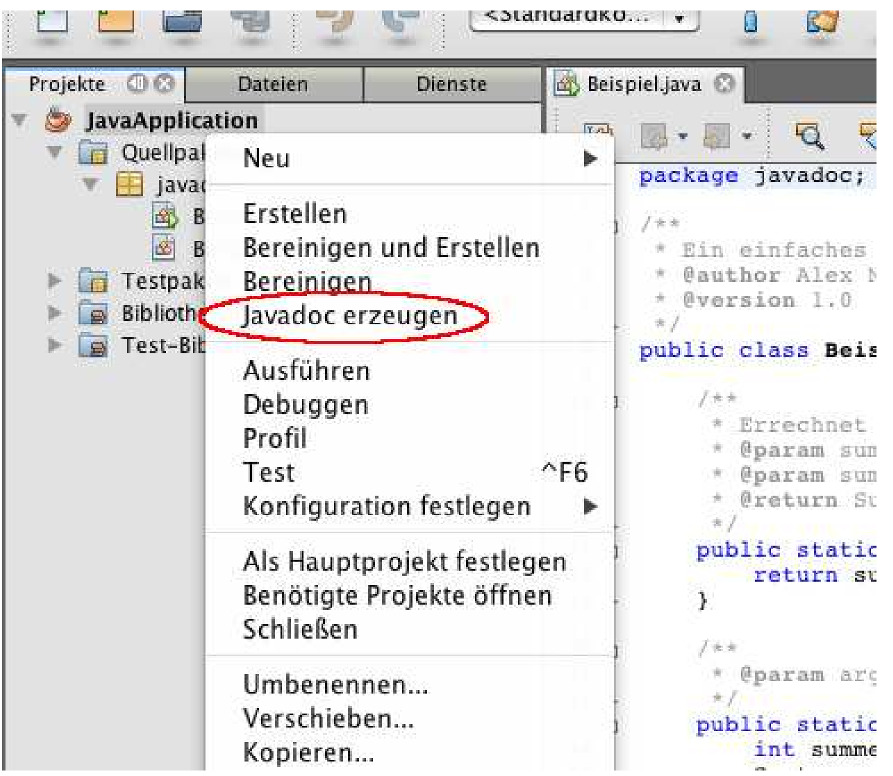
Der Befehl, um Javadoc auszuführen hat folgendes Schema:

```
javadoc [Optionen] Quelldatei(en)
```

`javadoc` : Das Programm (Javadoc)
`[Optionen]` : Optionale Parameter
`Quelldateien` : Angabe der zu dokumentierenden Datei(en)

Optionen (Auszug):
 `d -Verzeichnis-` : Verzeichnis, in das die Dokumentation gespeichert wird
 `nodeprecated` : `@deprecated`-Tags werden ignoriert
 `verbose` : Detaillierte Ausgabe von Javadoc. Zeigt zum Beispiel Warnungen an

- *In Netbeans*
  Eine Javadoc-Dokumentation kann auch direkt aus Netbeans heraus erstellt werden, was um einiges einfacher ist, als dies in der Konsole zu tun. In Netbeans genügt ein Rechtsklick auf das Projekt im Projekt-Fenster und ein Klick auf „Javadoc erzeugen".



Anschließend wird automatisch eine Dokumentation erzeugt, im Verzeichnis „Doku" im Projektordner gespeichert und im Standardbrowser geöffnet.

11.3.3. Erzeugte Dateien (Auszug)

index.html	Hauptansicht bestehend aus zwei Frames
allclasses-frame.html	Linker Frame
paketname/package-summary.html	Rechter Frame
overview-tree.html	Baumstruktur der Klassen → Verdeutlichung der Vererbung
deprecated-list.html	Übersicht über veraltete Methoden
help-doc.html	Kurze Beschreibung von Javadoc

paketname/Klassenname.html	Detaillierte Dokumentation der Klassen
stylesheet.css	Format für alle HTML-Dateien der Dokumentation

11.4. Erweiterbarkeit

Javadoc ist modular aufgebaut. Das heißt, es können Teile von Javadoc durch andere Teile ersetzt werden. Somit ist es möglich, dass beispielsweise der Tag-Wortschatz durch so genannte „Taglets" erweitert werden kann.
Die Ausgabe einer von Javadoc erstellten Dokumentation wird von so genannten „Doclets" erstellt. Standardmäßig wird ein HTML-Doclet verwendet. Es können jedoch zusätzliche Doclets installiert werden für andere Ausgabeformate, wie zum Beispiel XML oder PDF.

11.5. Was leistet Javadoc nicht?

Javadoc ist nicht geeignet für das Erstellen einer Dokumentation der Benutzerschnittstelle (siehe 1.2). Somit ist eine Javadoc-Dokumentation ausschließlich für Programmierer von Bedeutung und für den Anwender einer Software nutzlos bzw. unsichtbar.
Außerdem ist Javadoc nicht geeignet für den Entwurf von Software. Denn Software wird vor dem programmieren entworfen und Javadoc dokumentiert die Software nach (bzw. während) des Programmierens. Für den Entwurf wären zum Beispiel UML-Diagramme besser geeignet.

11.6. Quellen

Ullenboom, Christian: Java ist auch eine Insel (8. Auflage), Galileo Computing, 2009
http://de.wikipedia.org/wiki/Javadoc, November 2010
http://de.wikipedia.org/wiki/Softwaredokumentation, November 2010
http://www.imn.htwk-leipzig.de/~weicker/pmwiki/pmwiki.php/Main/Javadoc, November 2010

12. NetBeans GUI Editor – Artur Schmidt, 06.12.2010

12.1. Einleitung

Das Graphical User Interface (kurz GUI, deutsch: graphische Benutzeroberfläche) ist bei heutigen Software Anwendungen Standard und ist nicht mehr weg zu denken. Da GUI graphische (Fenster, Buttons u.s.w.) und Bedienungselemente (Unterstützung von Maus und Tastatur) enthält, gestaltet sich die manuelle Entwicklung eines GUI's sehr schwierig. Es erfordert umfangreiche Kenntnisse in der Programmierung. Daher bietet NetBeans einen GUI-Editor, mit welchem man per "drag and drop" Prinzip schnell und ohne Kenntnisse in Swing (NetBeans API für GUI) Programmierung, die Möglichkeit hat tolle Benutzeroberflächen zu gestalten und sofort zu nutzen.

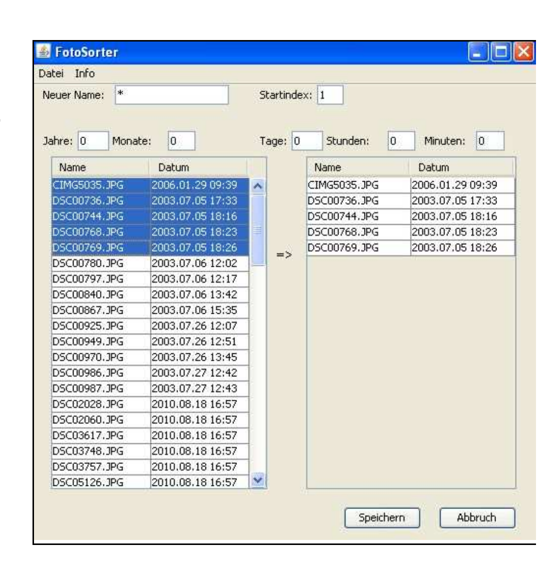


Abb. 43: Beispiel einer graphischen Benutzeroberfläche

Der NetBeans GUI-Editor benutzt Swing für die Erzeugung der Benutzeroberfläche. Selbst Swing basiert auf seinen Vorgänger AWT und benutzt immer noch dessen EventHandling, welches für die Bedienung der graphischen Benutzeroberfläche zuständig ist. Im Vergleich zu AWT bietet Swing mehr Objekte(Buttons, Panels u.s.w.) und eine umfangreichere Konfiguration der Objekte zur Auswahl.

12.2. Erstellung einer graphischen Benutzeroberfläche

Erstellen Sie einen JFrame in Ihrem Projekt, in dem Sie im Projektfenster auf Projekt dann Rechtsklick->New->JFrame-form eine neue Klasse hinzufügen. Nach der Benennung der Klasse und der Zuweisung einem Package können Sie mit der Gestaltung Ihrer Oberfläche sofort beginnen.

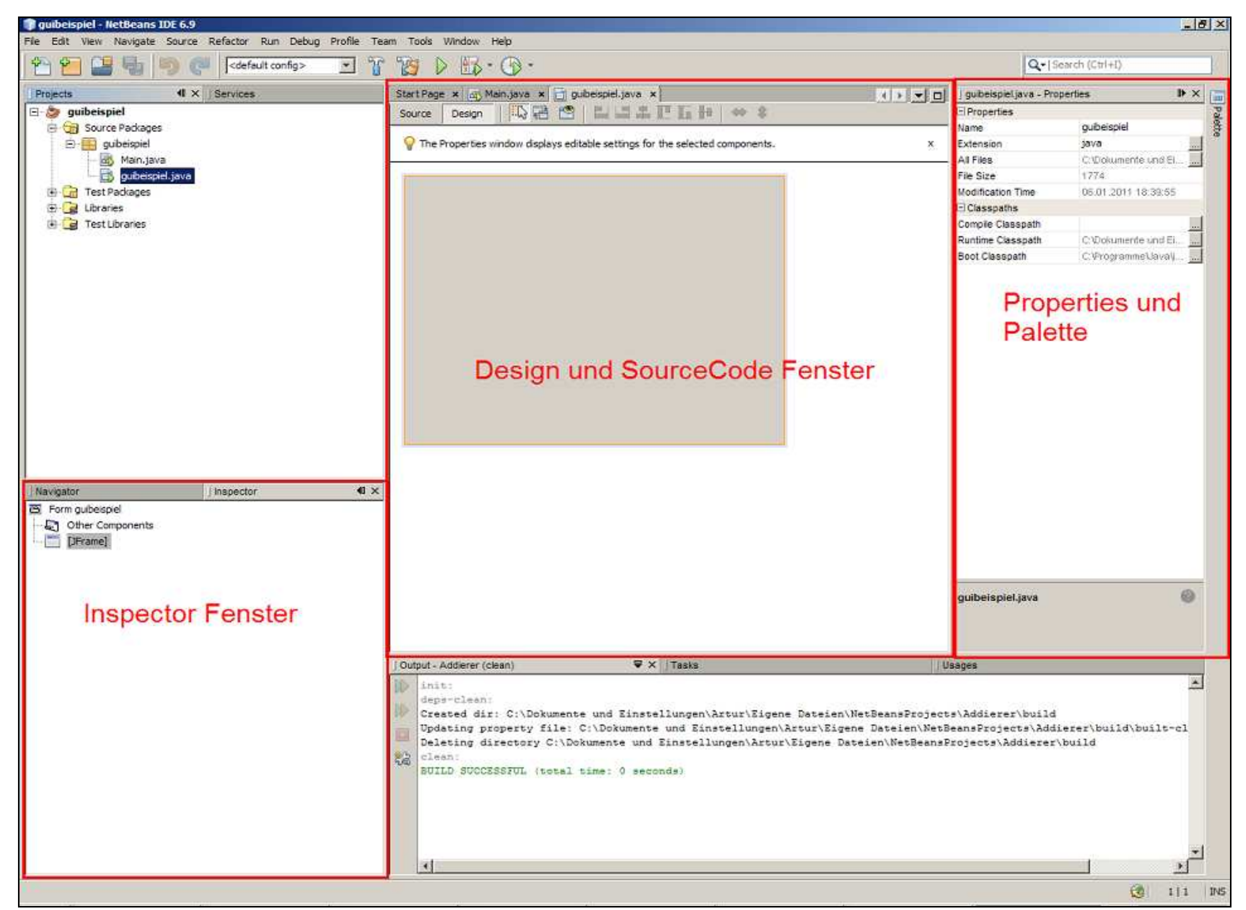


Abb. 44: Wichtige Fenster zur Erstellung der graphischen Benutzeroberfläche (rot markiert)

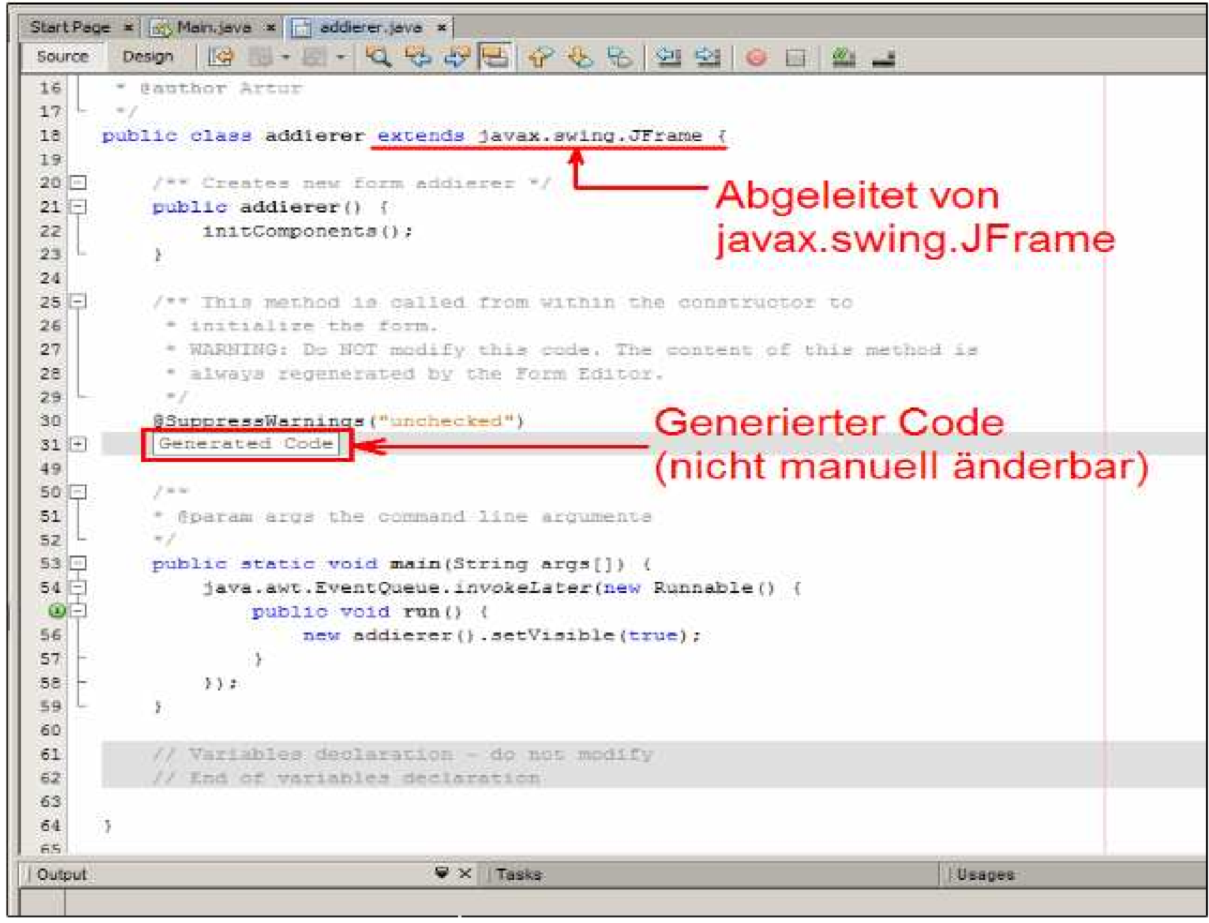


Abb. 45: Das vom Editor erzeugte Code

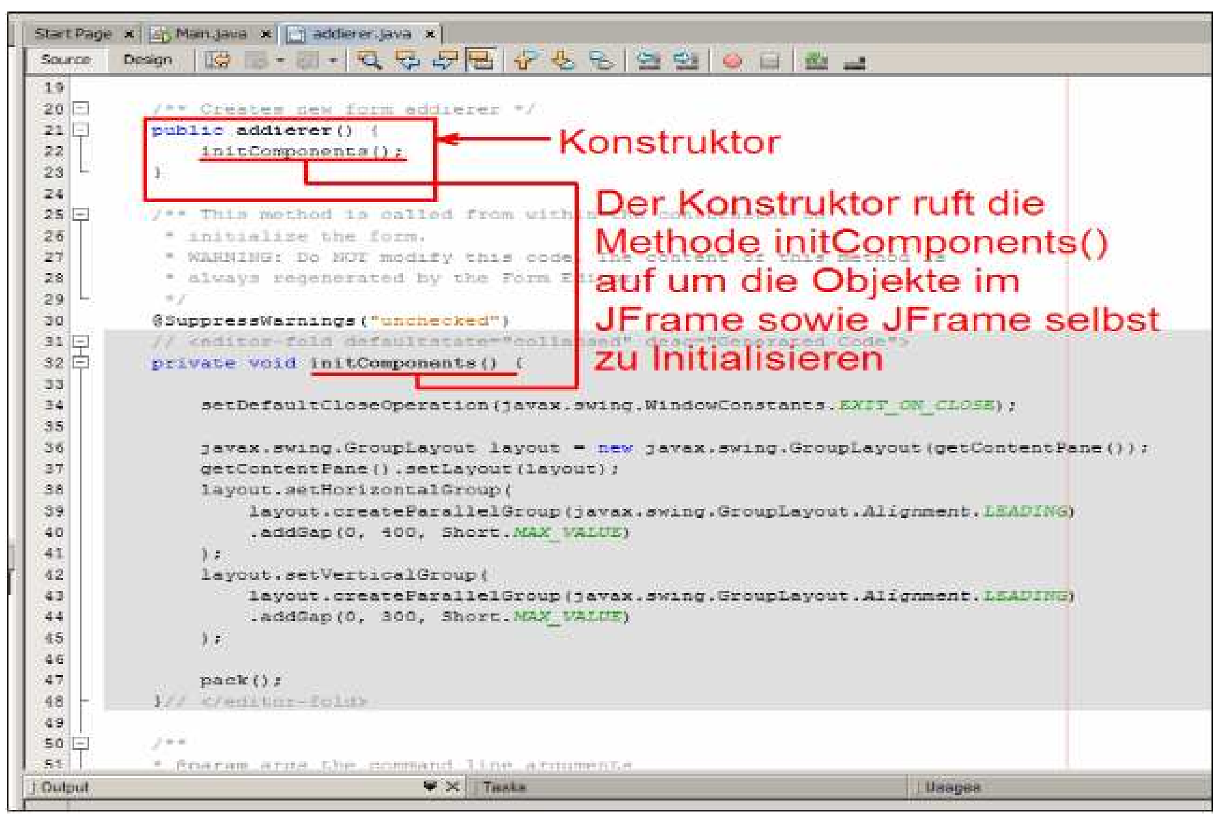


Abb. 46: SourceCode mit Eigenschaften der Objekte (grau markiert). Kann nicht editiert werden.

Das Design und SourceCode Fenster ist so gesehen Ihr Arbeitstisch auf dem Sie Ihren GUI erstellen. Die im Fenster existierendes GUI(grau in der Abb.2) und der SourceCode (siehe Abb.3 und 4) werden schon bei der Erstellung des JFrames vom Editor automatisch erzeugt. Als Standard können die Eigenschaften(Position, Backgroundcolor u.s.w.) der Objekte im SourceCode nicht manuell editiert werden. Der Editor wird den Code selber dynamisch ergänzen oder ändern falls Sie etwas im Design Modus ändern. Dadurch übernimmt der Editor die meiste Arbeit und der Entwickler kann sich somit auf das Design des GUI konzentrieren.

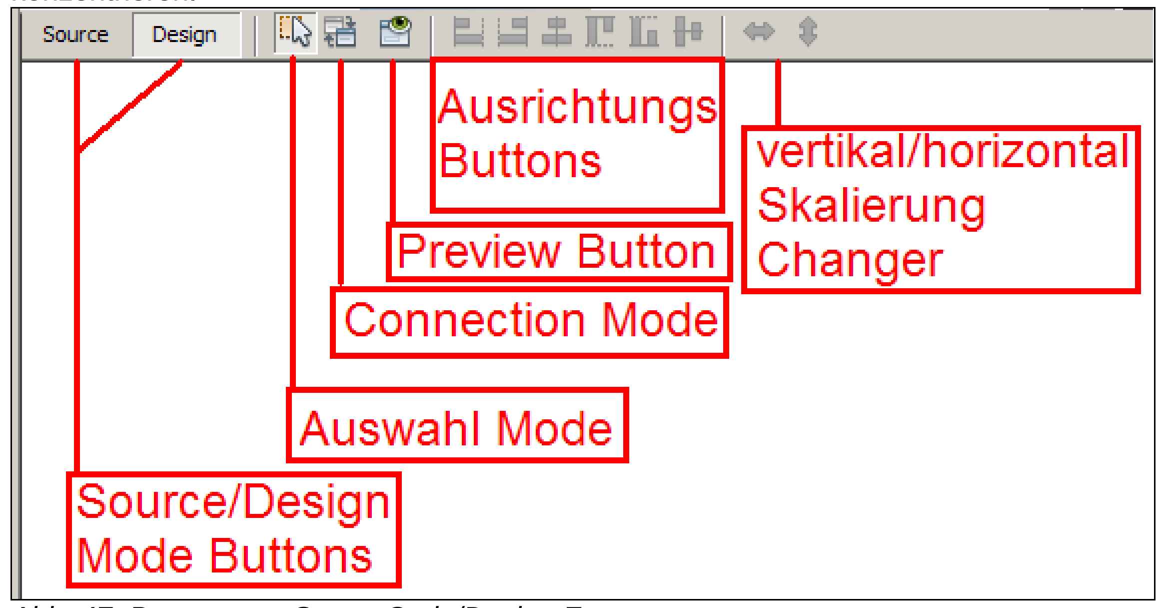


Abb. 47: Buttons von SourceCode/Design Fenster

- Source/Design Buttons: damit wechseln Sie zwischen SorceCode/Design Ansicht
- Auswahl Mode: ermöglicht das Auswählen von einzelnen/mehreren Objekten

- Connection Mode: verbindet Objekte miteinander für einen Event
- Preview Button: ermöglicht eine Vorschau des GUI
- Ausrichtungs Buttons: richtet ausgewählte Objekte aus
- vertikal/horizontal Skalierung Changer: setzt 2 horizontal/vertikal Anker und ermöglicht somit das Skalieren von Objekten relativ zur Fenstergröße

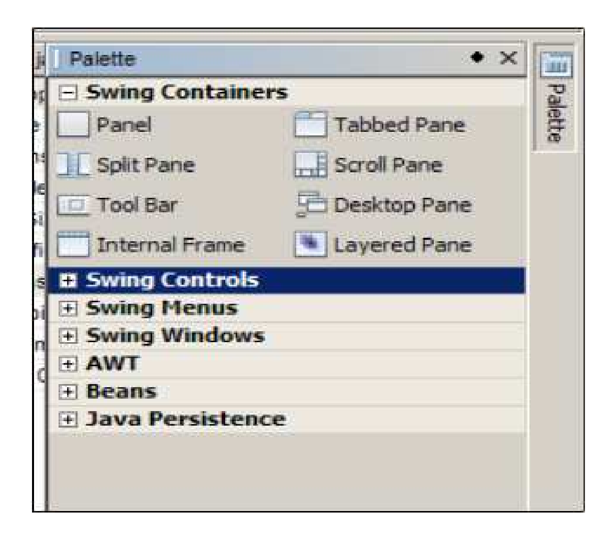
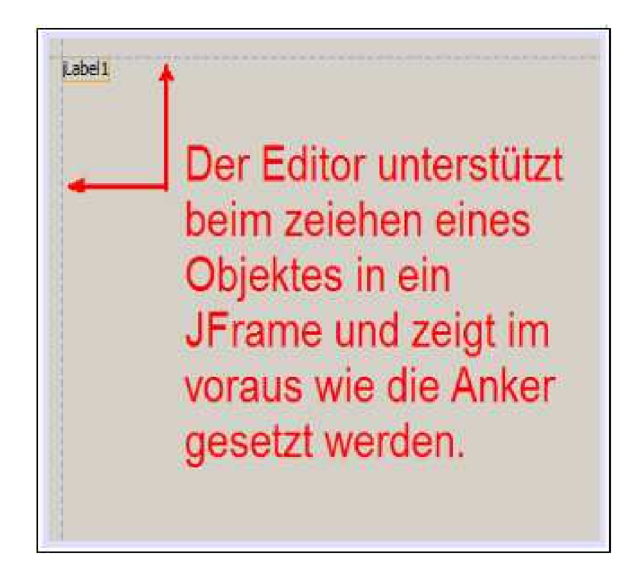


Abb. 48: Palette mit Objekten



Positionierung des Objektes

```
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents() {

    jLabel1 = new javax.swing.JLabel();

    setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);

    jLabel1.setText("jLabel1");

    javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
    getContentPane().setLayout(layout);
    layout.setHorizontalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
            .addGap(38, 38, 38)
            .addComponent(jLabel1)
            .addContainerGap(328, Short.MAX_VALUE))
    );
    layout.setVerticalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
            .addGap(39, 39, 39)
            .addComponent(jLabel1)
            .addContainerGap(247, Short.MAX_VALUE))
    );

    pack();
}// </editor-fold>

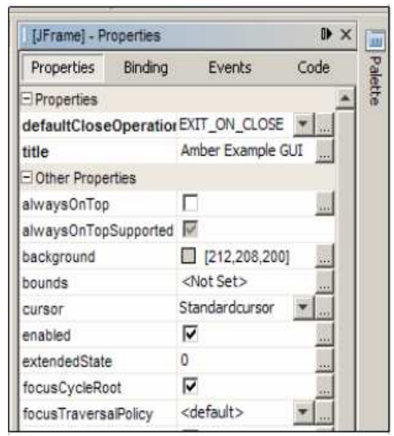
    // Variables declaration - do not modify
    private javax.swing.JLabel jLabel1;
    // End of variables declaration
```

Abb. 49: Neues Objekt im Code (rot markiert)

Die Palette beinhaltet alle Objekte die Ihnen Swing zur Verfügung stellt. Um einen Objekt dem GUI hinzuzufügen reicht einfach das gewünschte Objekt aus der Palette in die zu erstellende GUI zu ziehen "drag and drop". Beim Positionieren eines Objektes greift Ihnen der Editor wieder unter die Arme, so zeigt er Ihnen bestmögliche Platzierung der Anker und passt automatisch das Objekt und die Schrift auf die gleiche Ebene wie die der anderen Objekten, falls sich diese in der gleichen horizontalen/vertikalen Ebene befinden.

Die Anker(in NetBeans engl. Anchor) spielen bei der GUI eine wichtige Rolle. Sie bestimmen die Positionierung der Objekte, in dem der Abstand zwischen dem Objekt und dem Rand/Objekt, an dem das Objekt verankert ist, auch bei Vergrößerung des GUI eingehalten wird. Das heißt, wird ein Objekt Top/Left am Rand des GUI verankert, so wird dieser seine Position bei Vergrößerung des GUI oben-links einhalten.

Nach dem Hinzufügen eines neuen Objektes ins GUI ergänzt, wie oben schon erwähnt, der Editor den SourceCode mit dem Code des neu hinzugefügten Objektes.



Properties

Die Properties ermöglichen das Setzen und Verändern der Variablen/Eigenschaften einzelner Objekte. Der Editor wird jede Änderung dynamisch im SourceCode ändern.

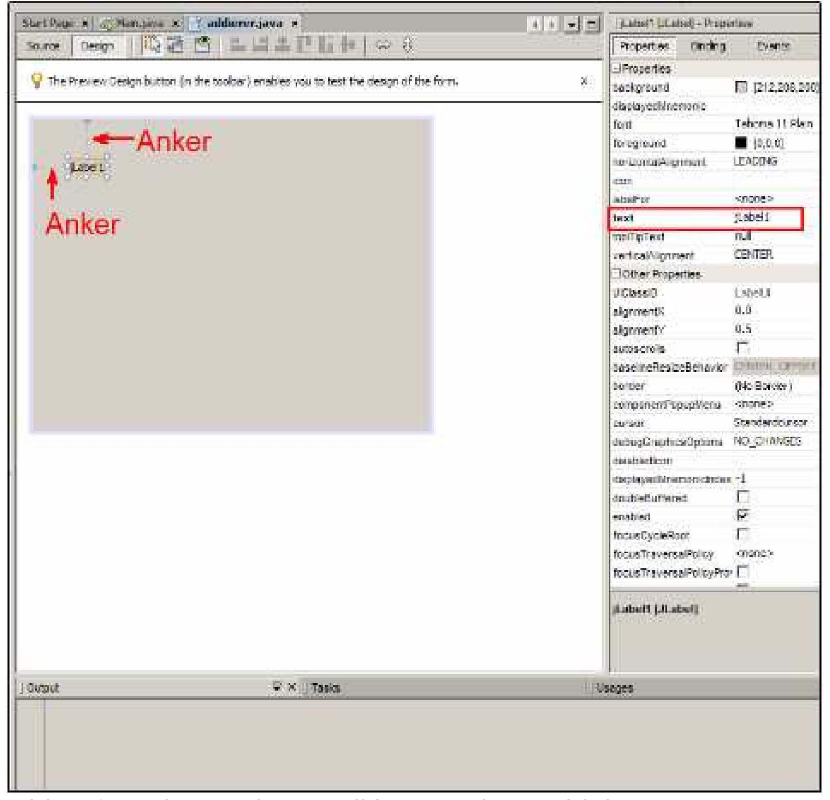


Abb. 50: Anker und Texteditierung eines Objektes

Beispiel: Um den Text von jLabel1 zu editieren, wählt man das Objekt aus und in Properties den Text (siehe Abb. rot markiert) editieren. Es ist auch möglich das Objekt direkt im Design-Fenster oder im Inspektor-Fenster per Rechtsklick->Edit Text zu editieren.

Das Navigator-Fenster dient der Übersicht und Verwaltung der hierarchischen Anordnung der Objekte. Es ist auch von hier aus möglich neue Objekte und Events zu erstellen oder auch die Properties zu editieren. Die Positionierung der Objekte muss jedoch im Design-Fenster unternommen werden.

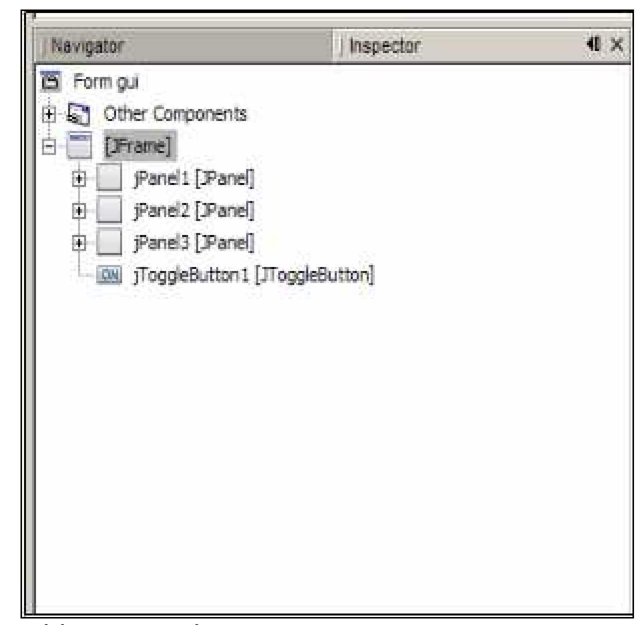


Abb. 51: Navigator-Fenster

12.3. EventHandler

Auch die schönste graphische Benutzeroberfläche ist nutzlos falls diese keine Funktionalitäten/Bedienung besitzt. Hier kommt der EventHandler (deutsch: Event Verwalter) welches für die Funktionalität(Eingabe per Tastatur oder Mausklicks) einer GUI sorgt. Ein Event kann jedem Objekt, über Rechtsklick->Events->und Event nach Auswahl wählen, zugewiesen werden. Der Editor erstellt dann den gewünschten Event automatisch. Tritt der Event ein, welches einem Objekt zugewiesen wurde, wird dieser von dem EventListener(deutsch eher: Event Beobachter) abgefangen/registriert und leitet diesen an EventHandler weiter. Nachdem ein Event für ein oder mehrere Objekte erstellt wurde, wechselt der Editor automatisch von Design in die Source Ansicht genau zu der Stelle im Code, wo der Event vom Editor erstellt wurde(siehe Abb. rot markiert), da die Anweisungen im Event "noch" manuell editiert werden müssen.



Abb. 52: Erstelltes Event. Der Code der das Event ausführt muss manuell editiert werden

12.4. Quellen

http://download.oracle.com/javase/tutorial
http://www.thenewboston.com/
http://wiki.netbeans.org/UsingGUIEditor
http://www.youtube.com/watch?v=LFr06ZKIpSM

13. GUI-Editor II (Beans & Binding) - Vladimir Komarov, 06.12.2010

13.1. Beans in NetBeans

Beans sind eigene benutzerdefinierte Komponente in NetBeans. Um ein Beans zu erstellen, reicht es schon Klasse zu kompilieren und über rechte Maustaste zur Palette hinzufügen. Zusätzlich kann man noch über speziellen BeanEditor Beans Eigenschaften anpassen. Wobei NetBeans automatisch die getter/setter Methoden, Properties und Events erkennt.

Schritt 1: Quelldatei
Praktisch reicht es schon eine Klasse mit einer **Variable** und **get/set Methoden,** um aus diesem Quelldatei ein **Bean** zu erstellen.

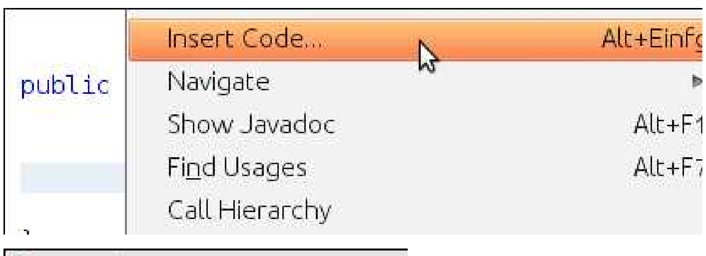
```
public class NewClass {
    String beispiel2="b2";

    public String getBeispiel2() {
        return beispiel2;
    }

    public void setBeispiel2(String beispiel2) {
        this.beispiel2 = beispiel2;
    }

}
```

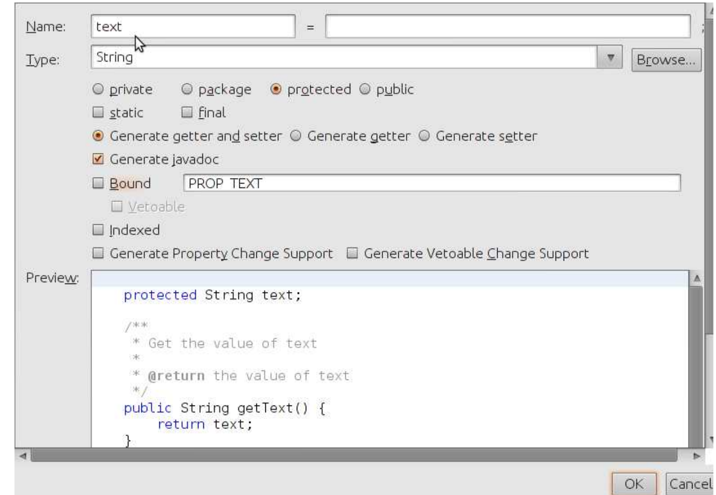
Dieser Code kann man auch automatisch generieren lassen. Dazu rechte Maustaste im Quelltext auf dem gewünschten Platz klicken. Im geöffneten PopUp Menü, klickt man auf **Insert Code...** und im nächsten Menü auf **Add Property...**



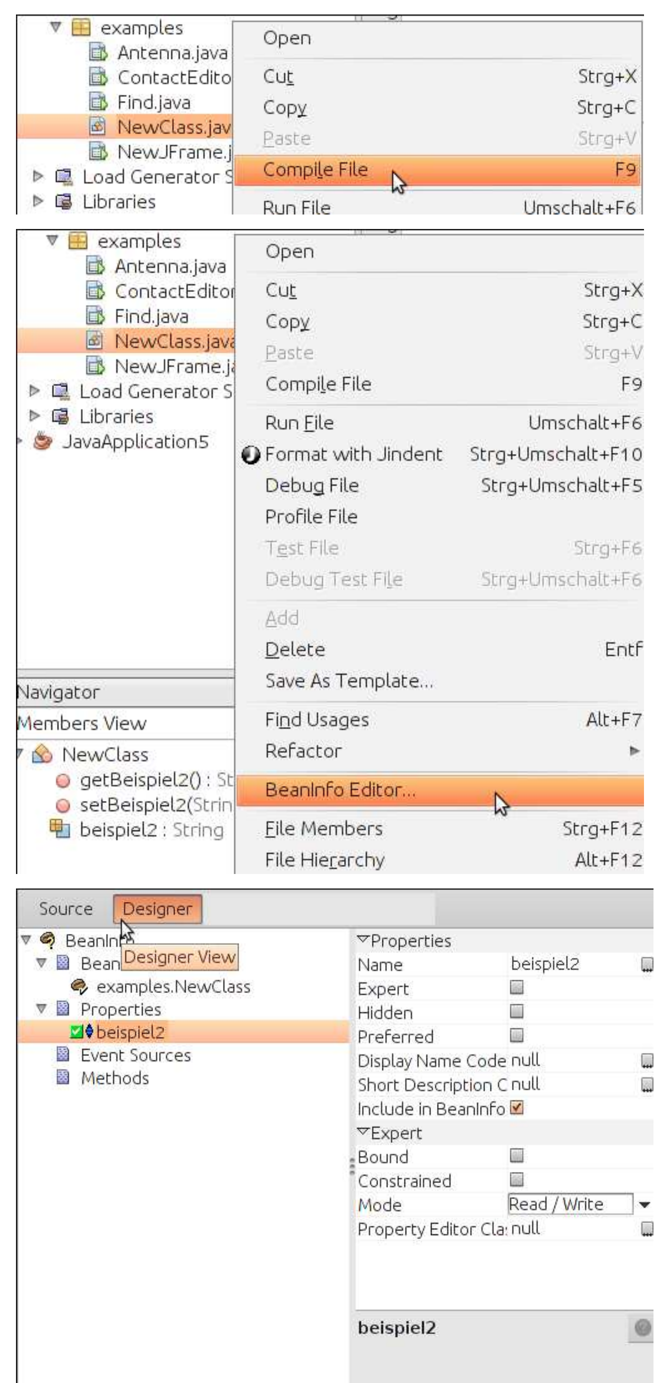




Unter **Name** wird Variablenname eingegeben, die erstellt werden soll. Im Eingabefeld nach dem **Gleichheitszeichen** können Initialisierungswert, andere Variable oder Funktionsname eingegeben werden.
Auch Variablentyp kann unter **Type** ausgewählt werden.
Es können Get, Set oder beide Methoden generiert werden.
Eine wichtige Option ist **Bound**. Wenn diese Option aktiviert ist, werden alle GUI-Komponente, die mit dieser Variable über **Binding** verbunden sind, **automatisch synchronisiert,** auch wenn man diese **Get/Set Methoden** innerhalb seines Quelltext aufruft. Dazu wird ein **PropertyChangeListener** definiert, der alle Aufrufe überwacht und andere Komponente benachrichtigt.
Indexed: erstellt ein Array Variable, anstatt



einfache Variable.



Schritt 2: Klasse kompilieren (falls noch nicht kompiliert)
Rechts-klick auf zu kompilierende Klassdatei und dann auf **Compile File** oder mit **F9**

Schritt 3: BeanInfo erstellen
Rechts-klick auf Klassdatei und dann auf **BeanInfo Editor...**
Es öffnet sich eine Abfrage, ob BeanInfo neu erstellt werden soll und nach dem Klick auf **Yes** erscheint Quelldatei von dem BeanInfo.



Schritt 4: Bean anpassen
Wenn man von **Source** auf **Designer** Editor umschaltet, kann man bequem Eigenschaften, die man zur Benutzung freigeben möchte, aktivieren (im Bild – Markiertes Element **beispiel2**) oder deaktivieren. Dafür aktiviert oder deaktiviert man Option **Include in BeanInfo**

Schritt 5: Bean zur Palette hinzufügen
Bean soll auf der GUI-Palette vorhanden sein. Dazu klickt man wieder mit der rechten Maustaste auf unsere Klassdatei und in geöffnetem PopUp-Menü auf **Tools** und dann auf **Add to Palette...**
Zum Schluss wird Kategorie abgefragt, unter welcher unserer Element zugeordnet wird (zweites Bild). NetBeans hält extra Kategorie für selbst erstellte Beans parat, die kaum zu glauben, Beans heißt.

Jetzt kann man unseren Bean wie jeden anderen GUI-Element benutzen, vorausgesetzt – der Pfad zur kompilierter Datei ist im aktuellen Projekt eingetragen.

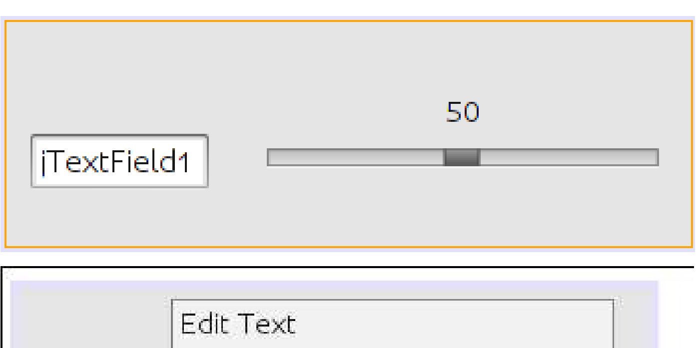
13.2. Binding in NetBeans

In der Regel ist die Entwicklung von GUI-Programmen von Anfang an an den GUI-Editor gebunden. Schon bei der Funktionen-Entwicklung muss man sich Gedanken machen, über die Vorgehensweise, wie die „in-out“ Funktionswerte an GUI angepasst werden sollen und

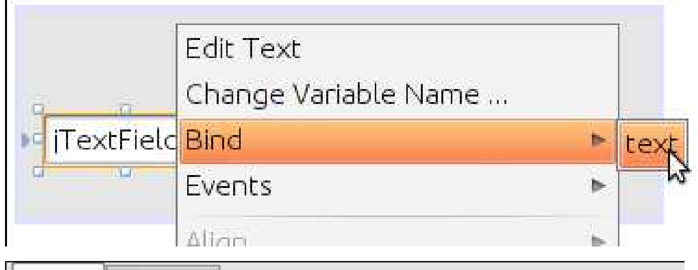
welche Typumwandelungen notwendig sind. Das bedeutet, das Programm und GUI „zusammenwachsen“ und man kann diese nicht trennen und z.B. Quellcode für Anderes GUI verwenden.
Binding in Netbeans ermöglicht fast automatische Anbindung von GUI Elementen an andere GUI Elemente, Variablen oder Funktionen(Properties). Sogar Synchronisierung zwischen GUI und dem Wert einer Variable kann mit wenigen Klicks automatisiert werden.
Am einfachsten ist das Verbinden von GUI Elementen. Hier können z.B. Text-Field mit einem Schieber direkt über GUI-Editor verbunden werden.

13.2.1. Zwei GUI-Elemente verbinden.

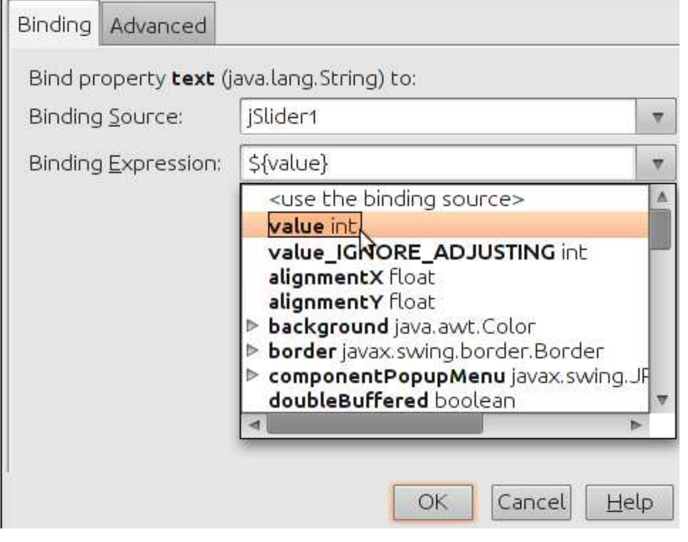
Schritt 1: Ausgangssituation
Textfeld = jTextField1
Schieber = jSlider1



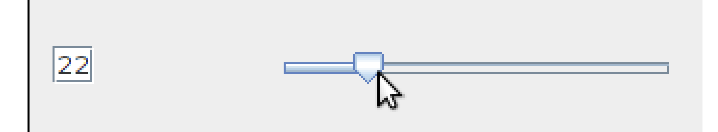
Schritt 2: Eigenschaft auswählen
Beim klicken mit der rechten Maus Taste auf Textfeld erscheint PopUp-Menü. In dem Menü geht man auf **Bind** und dann auf gewünschte Eingenschaft. In unserem Fall ist das nur eine Eigenschaft: **text**.



Schritt 3: Elemente verbinden
Im geöffneten Fenster wird Binding verhalten angepasst. Als **Binding Source** wählen wir unseren Schieber jSlider1. Und unter **Binding** Expression entsprechende Eingenschaft von dem Schieber.
Zu beachten: Text im Textfeld ist vom Typ **Sting** und ausgewählte Eigenschaft vom Schieber ist von Typ **int**. Die Umwandlung passiert hier im Hintergrund. Wir hätten auch Text-Feld an Schieber anbinden können, aber dann hätten wir einen „Umwandler“ gebraucht, um int-Werte in Text zu konvertieren.



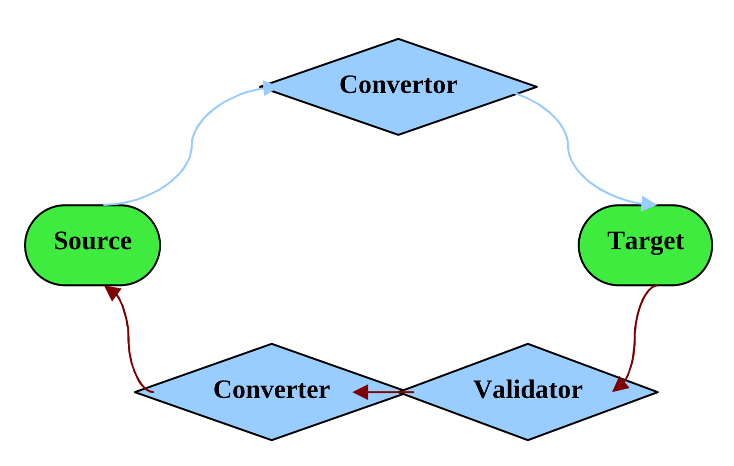
Schritt 4: Testen des Programm
Nach dem Starten erkennt man, dass verschieben des Schiebers den Text ändert oder auch Eingabe von Text den Schieber bewegt.



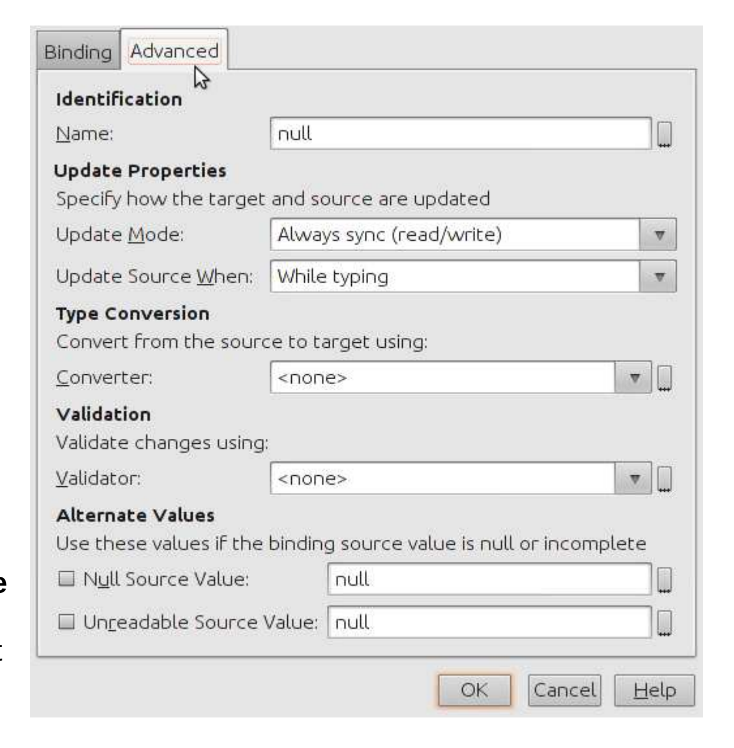
13.2.2. Datentypen umwandeln.

Wie im Abschnitt davor beschrieben, kann man auch eine **String Variable** und GUI-Element mit **Integer Eigenschaft** verbinden. Dazu hält NetBeans einige Einstellungen parat. Diese Einstellungen erreicht man unter **Advanced** Tab im Fenster für **Binding**.

Um unterschiedlichen **Datentypen zu synchronisieren,** können drei Funktionen eingebunden werden.
Convertor: wandelt die Eingegebene Daten in einen anderen Datentyp um und Zurück.
Validator: prüft eingegebenen Wert auf Konsistenz. Mit dem Validator können Eingabefehler erkannt und korrigiert werden.
Zwischen **Source** und **Target** wird kein Validator benutzt, weil NetBeans davon ausgeht, dass unsere Klasse schon richtigen Typ liefert.



Advanced Tab beim Binding



Update Mode: konfiguriert - wann synchronisiert wird. Dazu müssen auch geeignete Methoden vorhanden sein (Set/Get Methoden)

Update Source When: wann soll synchronisiert werden.

Converter: hier wird unsere Konvertierungsfunktion eingetragen.

Validator: Funktion zum Prüfen von Eingaben.

Beim Null Source Value und **Unreadable Source Value** kann man natürlich Defaultwert eingeben, falls Eingaben nicht korrekt waren.

13.2.3. Variable mit GUI-Element verbinden.

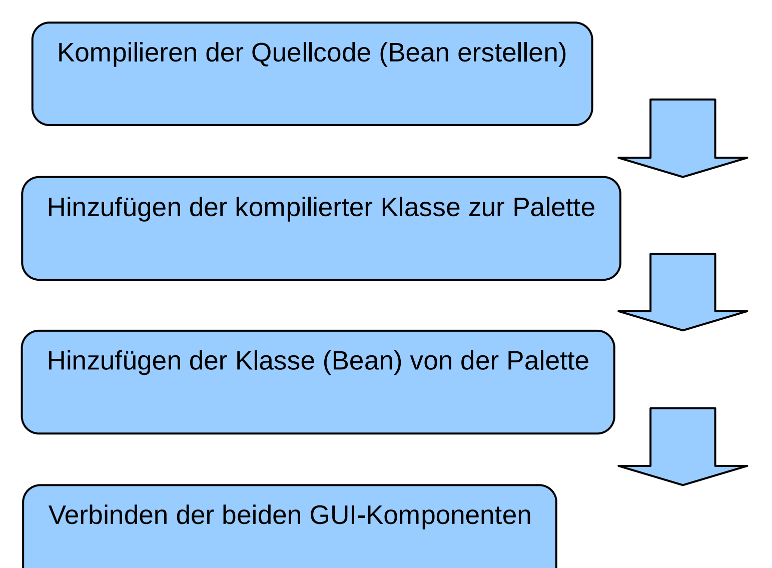
Beim Verbinden einer Variable mit einem GUI-Element ist mehr Aufwand erforderlich. Dazu muss man ein Bean aus dem Quelldatei erstellen.

1. Klassen-Datei muss kompiliert sein, erst dann kann man diese Klasse zur Palette hinzufügen.

2. Klasse wird zur Palette hin gefügt und erscheint da als benutzerdefiniertes GUI-Element.

3. Jetzt kann dieser GUI-Element wie jeder anderer Element zu der Form hin gefügt werden.

4. Da jetzt unsere Klasse wie normaler GUI-Element verhält, können wir diesen Element mit jedem anderen Standard-GUI-Element verbinden.



Quellen:

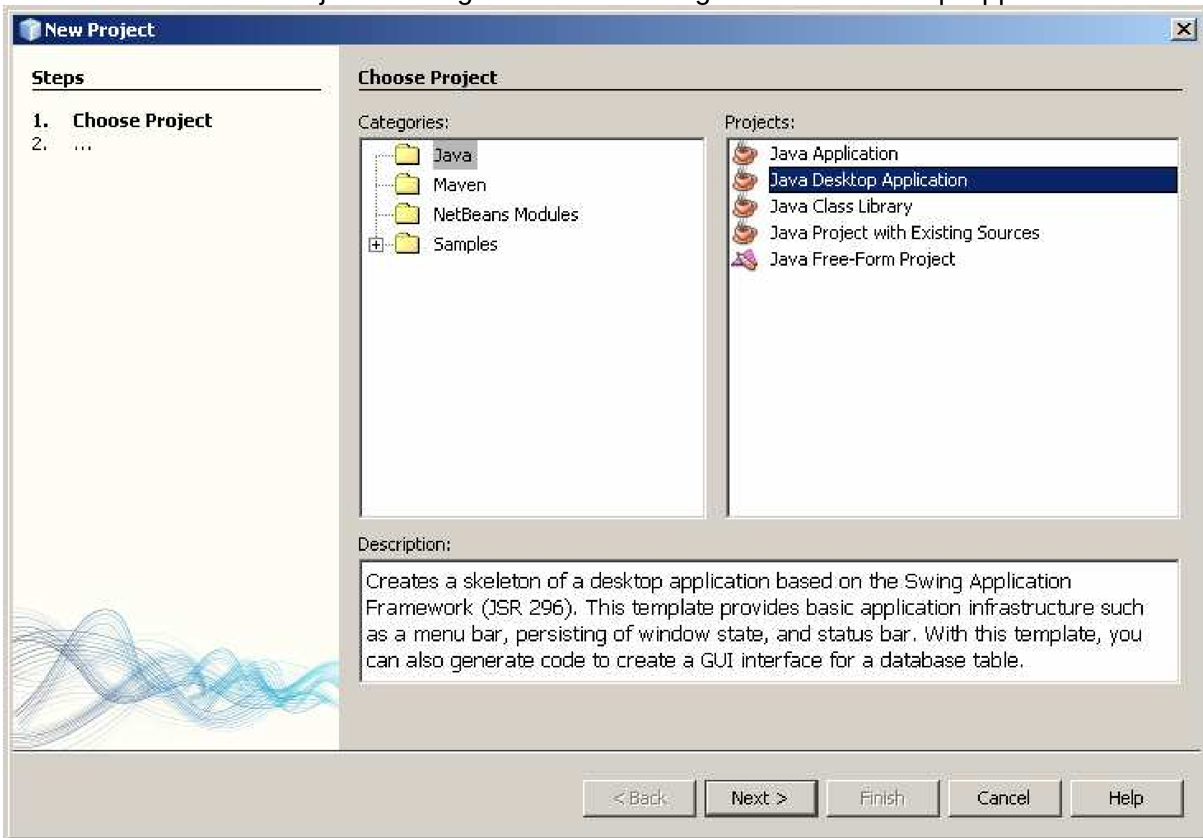
http://de.wikipedia.org/wiki/Netbeans
http://de.wikipedia.org/wiki/JavaBeans
http://download.oracle.com/javase/tutorial/javabeans/TOC.html
http://wiki.netbeans.org/CreatingImageBeanUsingNetBeans6.1
http://netbeans.org/kb/docs/java/gui-binding.html
http://openbook.galileocomputing.de/javainsel8/javainsel_07_004.htm
http://netbeans.org/download/flash/netbeans_6_gui_builder/netbeans_6_gui_builder.html

14. Java Desktop Application

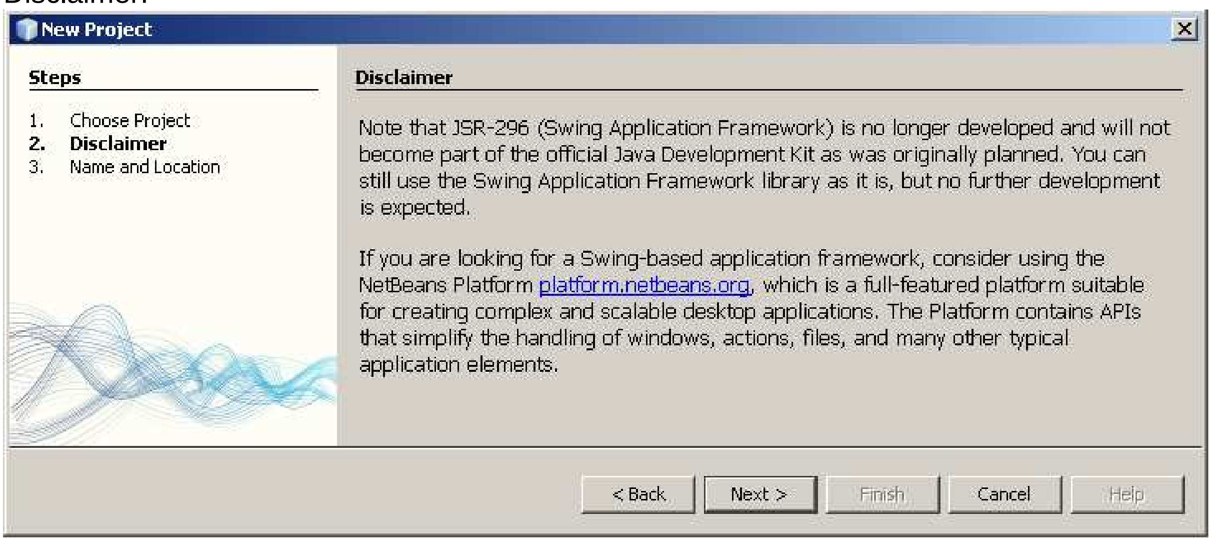
Für die Erstellung von kleineren Anwendungen wurde in den letzen Jahren das Java-Framework JSR 296 entwickelt. Dieses Framework wird voll von Netbeans unterstützt und soll deshalb hier auch kurz vorgestellt werden.

14.1. Einleitung

Der Wizard für neue Projekte ermöglicht die Erstellung von Java-Desktop Applications:



Da aber die JSR 296 nicht mehr weiter entwickelt wird, erscheint seit Netbeans 6.9 folgender Disclaimer:



Hier wird bereits auf Rich Client Entwicklung hingewiesen, was allerdings aufgrund des Umfangs nicht Gegenstand des Seminars war.
Im nächsten Schritt stellt der Wizard dann 2 Möglichkeiten zur Verfügung:

New Desktop Application

Steps

1. Choose Project
2. Disclaimer
3. **Name and Location**

Name and Location

Project Name: DesktopApplication1

Project Location: D:\Daten\Ablagen\NetBeansProjects Browse...

Project Folder: D:\Daten\Ablagen\NetBeansProjects\DesktopApplication1

Application Class: desktopapplication1.DesktopApplication1

Choose Application Shell

Basic Application
Database Application

A basic application skeleton containing everything needed to start with a general desktop application, such as the following:

- main frame with a menu bar and status bar
- About box
- main `Application` class
- implementation of status bar messages and progress indication
- resource management mechanisms to better handle loading of icons and strings

Use Dedicated Folder for Storing Libraries

Libraries Folder: Browse...

Different users and projects can share the same compilation libraries (see Help for details).

Set as Main Project

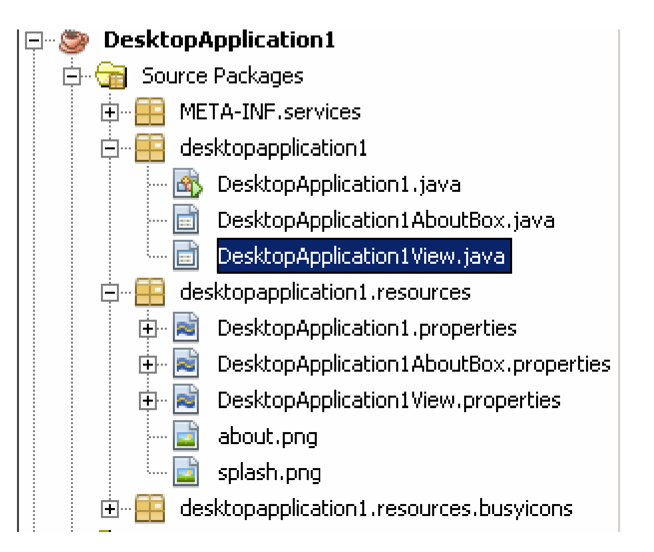
< Back Next > Finish Cancel Help

- Basic Application
- Database Application

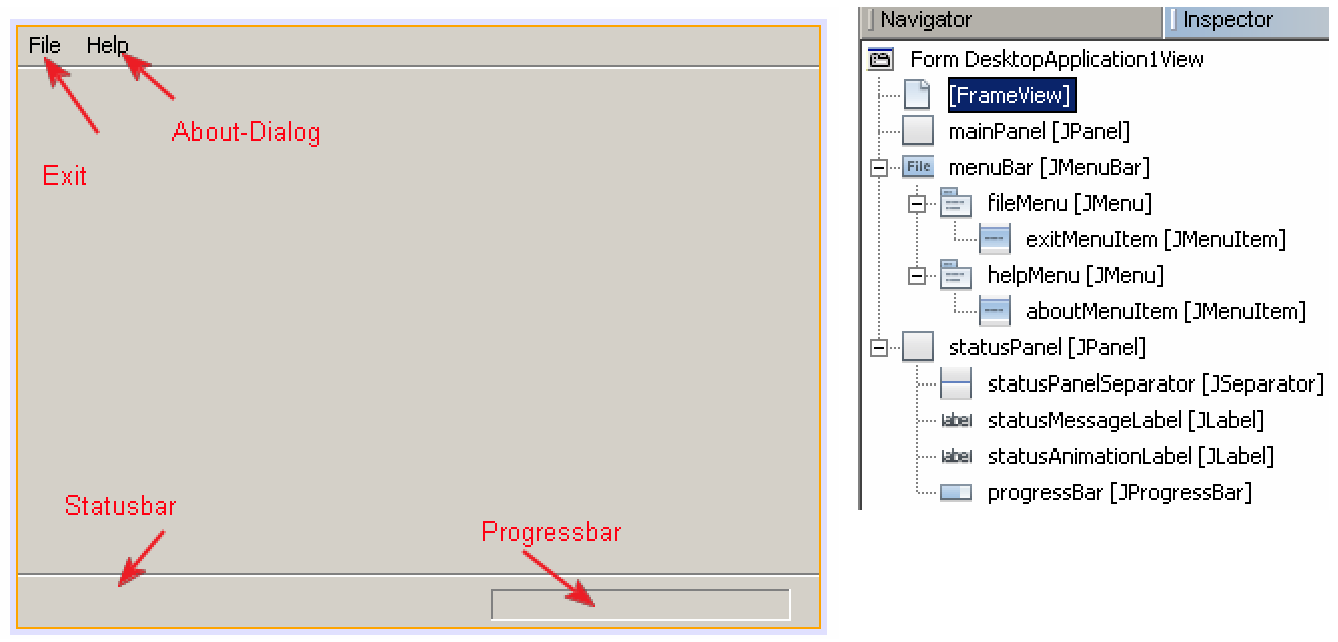
Im ersten Schritt soll das Thema Basic Application kurz erläutert werden.

14.2. Basic Application

Im Sourcepaket des Projektes erzeugt Netbeans nach Fertigstellen des Wizard 3 Klassen. Die *DesktopApplication1.java* ist eine Erweiterung der Klasse SingleFrameApplication und enthält Start und Initialisierung. Die grafische Oberfläche wird in *DesktopApplication1View.java* und der „Über“-Dialog in *DektopApplication1AboutBox.java* erzeugt.



Im Editor öffnet sich der GUI-View und zeigt, dass bereits wesentliche Elemente einer Anwendung zur Verfügung stehen.



GUI-View mit zugehörigem Inspector, wie es der Wizard komplett erzeugt.

Folgende beiden Links geben einen sehr guten Einblick in die JSR 296:
https://appframework.dev.java.net/intro/index.html
http://java.sun.com/developer/technicalArticles/javase/swingappfr/
Hier sollen nur kurz ergänzende Hinweise erfolgen, um das Framework etwas besser verstehen zu können.

14.3. Erläuterungen zum erzeugten Code

Code des Hauptprogramms:

```
package desktopapplication1;

 import org.jdesktop.application.Application;
 import org.jdesktop.application.SingleFrameApplication;

public class DesktopApplication1 extends SingleFrameApplication
{

    @Override protected void startup() {
        show(new DesktopApplication1View(this));
    }

    public static DesktopApplication1 getApplication() {
        return  Application.getInstance(DesktopApplication1.class);
    }

    public static void main(String[] args) {
        launch(DesktopApplication1.class, args);
    }
}
```

startup(): Startet das GUI und übergibt als Referenz ein Appllication-Objekt.

getApplication(): Liefert die Application-Referenz als Singleton-Instance ‚jederzeit'.

main(): Startet das Programm und kümmert sich um den EDT.

Code der View-Klasse:
Der Konstruktor:

```
public DesktopApplication1View(SingleFrameApplication app) {
      super(app);

      initComponents();

      // status bar initialization - message timeout, idle icon and busy animation, etc
      ResourceMap resourceMap = getResourceMap();
```

Kennen wir von jedem JFrame, welches uns Netbeans erzeugt.
ResoureMap besorgen.
Zeitwert holen.

```
    int messageTimeout = resourceMap.getInteger("StatusBar.messageTimeout");
    messageTimer = new Timer(messageTimeout, new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            statusMessageLabel.setText("");
        }
    });
    messageTimer.setRepeats(false);
    int busyAnimationRate = resourceMap.getInteger("StatusBar.busyAnimationRate");
    for (int i = 0; i < busyIcons.length; i++) {
        busyIcons[i] = resourceMap.getIcon("StatusBar.busyIcons[" + i + "]");
    }
    busyIconTimer = new Timer(busyAnimationRate, new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            busyIconIndex = (busyIconIndex + 1) % busyIcons.length;
            statusAnimationLabel.setIcon(busyIcons[busyIconIndex]);
        }
    });
    idleIcon = resourceMap.getIcon("StatusBar.idleIcon");
    statusAnimationLabel.setIcon(idleIcon);
    progressBar.setVisible(false);

    // connecting action tasks to status bar via TaskMonitor
    TaskMonitor taskMonitor = new TaskMonitor(getApplication().getContext());
    taskMonitor.addPropertyChangeListener(new java.beans.PropertyChangeListener() {
        public void propertyChange(java.beans.PropertyChangeEvent evt) {
            String propertyName = evt.getPropertyName();
            if ("started".equals(propertyName)) {
                if (!busyIconTimer.isRunning()) {
                    statusAnimationLabel.setIcon(busyIcons[0]);
                    busyIconIndex = 0;
                    busyIconTimer.start();
                }
                progressBar.setVisible(true);
                progressBar.setIndeterminate(true);
            } else if ("done".equals(propertyName)) {
                busyIconTimer.stop();
                statusAnimationLabel.setIcon(idleIcon);
                progressBar.setVisible(false);
                progressBar.setValue(0);
            } else if ("message".equals(propertyName)) {
                String text = (String)(evt.getNewValue());
                statusMessageLabel.setText((text == null) ? "" : text);
                messageTimer.restart();
            } else if ("progress".equals(propertyName)) {
                int value = (Integer)(evt.getNewValue());
                progressBar.setVisible(true);
                progressBar.setIndeterminate(false);
                progressBar.setValue(value);
            }
        }
    });
}
```

Timer setzen, der MessageLable nach messageTimeOut löscht.

BusyIcons holen.

Im Takt busyAnimationRate das nächste Icon setzen.

Für Background-Tasks. An den Taskmonitor (ist Teil des FrameView) neuen Eigenschaftslistener hängen, der BusyIcon und Progress in Abhängigkeit des Status setzt.

About Box aufrufen:

```
@Action
public void showAboutBox() {
    if (aboutBox == null) {
        JFrame mainFrame = DesktopApplication1.getApplication().getMainFrame();
        aboutBox = new DesktopApplication1AboutBox(mainFrame);
        aboutBox.setLocationRelativeTo(mainFrame);
    }
    DesktopApplication1.getApplication().show(aboutBox);
}
```

Application besorgen. AboutDialog instanzieren.

Dialog anzeigen.

Rest:

```
@SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">
private void initComponents() {…}
```

Init der Komponenten, wie wir es vom GUI-Editor kennen

```
// Variables declaration - do not modify
private javax.swing.JPanel mainPanel;
private javax.swing.JMenuBar menuBar;
private javax.swing.JProgressBar progressBar;
private javax.swing.JLabel statusAnimationLabel;
private javax.swing.JLabel statusMessageLabel;
private javax.swing.JPanel statusPanel;
// End of variables declaration

private final Timer messageTimer;
private final Timer busyIconTimer;
private final Icon idleIcon;
private final Icon[] busyIcons = new Icon[15];
private int busyIconIndex = 0;

private JDialog aboutBox;
```

Variablen, die wir für einen Background-Tasks brauchen.

14.4. Der ExitListener

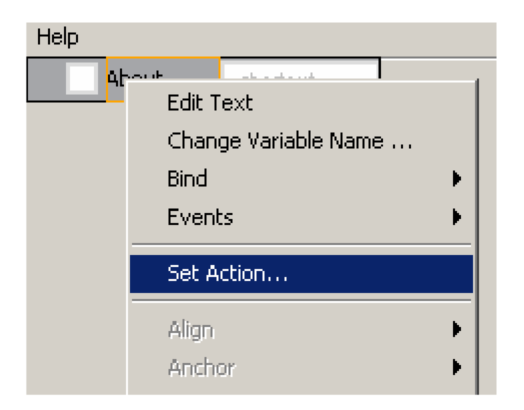
In den oben genannten Links wird vorgestellt, wie ein **ExitListener** zum Schließen des Programms erstellt werden kann. Bei der Aufteilung zwischen GUI und Application, wie es Netbeans für uns erledigt, sollte dieser dann wie folgt programmiert werden:

```
27  private class MyExitListener implements ExitListener {
28
29      public boolean canExit(EventObject e) {
30          boolean bOkToExit = false;
31          Component source = (Component) e.getSource();
32          bOkToExit = JOptionPane.showConfirmDialog(source,
33                  "Do you really want to exit?")== JOptionPane.YES_OPTION;
34          return bOkToExit;
35      }
36
37      public void willExit(EventObject event) {
38      }
39  }
40
41  public DesktopApplication1View(SingleFrameApplication app) {
42      super(app);
43
44      initComponents();
45
46      app.addExitListener(new MyExitListener());
47
```
………………

Auf diese Weise wird erreicht, dass auch weiterhin der GUI-Code in der View-Klasse bleibt.

14.5. Arbeiten mit dem Action-Dialog

Das Arbeiten mit einer ActionMap (liefert die JSR 296 automatisch mit) erleichtert dem Entwickler die Arbeit mit Actions, da für Menüs, Toolbars, Buttons und Tastaturbefehle nur eine Action definiert werden muss. In Netbeans wird dies sehr einfach unterstützt.



Klick mit der rechten Maustaste auf eine Komponente und dann über das Menü „Set Action“ eine Action der ActionMap hinzufügen.

Der folgende Dialog ermöglicht sehr einfach entsprechende Vorgaben zu machen (hier das Beispiel der Action „schowAboutBox“.

Set Action...
Set **aboutMenuItem**'s **action** property using: Default editor
Action: showAboutBox
Vorhandene Action auswählen oder "createNew" auswählen.
Action's Class: desktopapplication1.DesktopApplication1View
Action's Method: @Action void showAboutBox()
Background Task
ggfs. als Backgroundtasks laufen lassen
Attributes: Basic Advanced
Beschriftung
Text: About...
Tool Tip: Show the application's information dialog
Tooltip
Accelerator: Ctrl Shift Alt Meta (Mac only)
Tastaturkürzel
Letter: Clear
Icon: Small Icon... Large Icon...
Icons, wenn gewünscht
OK Reset to default Cancel Help

Wird eine neue Actionmethode angelegt, so wechselt Netbeans nach dem OK-Klick direkt im Editor zu dem Codegerüst:

```
201    @Action
202    public void goTest() {
203    }
204
205    // Variables declaration - do not modify
```

Normalerweise wird dieser Code unmittelbar vor die Variablendeklaration geschrieben. In manchen Fällen kann es sein, dass Netbeans nicht in den Codeeditor wechselt. In der Regel liegt dies daran, dass Netbeans nicht weiß, wo es den Code hinschreiben soll und dann gar keinen Code erzeugt. In der Regel hilft hier einige Methoden zu verschieben oder den Methodenrumpf von Hand zu schreiben und dann die Action im setAction-Dialog auszuwählen.
Wird ein Background-Task gewünscht (dieser ermöglicht weiter die Bedienung der Oberfläche, während im Hintergrund umfangreiche Berechnungen oder das Laden von Dateien durchgeführt werden), so ist einfach ein Haken im Set-Action-Dialog zu setzen. Netbeans erzeugt dann automatisch folgenden Code:

```
@Action
   public Task goTest() {
      return new GoTestTask(getApplication());
   }

   private class GoTestTask extends org.jdesktop.application.Task<Object, Void> {
      GoTestTask(org.jdesktop.application.Application app) {
         // Runs on the EDT.  Copy GUI state that
         // doInBackground() depends on from parameters
         // to GoTestTask fields, here.
         super(app);
      }
      @Override protected Object doInBackground() {
         // Your Task's code here.  This method runs
         // on a background thread, so don't reference
         // the Swing GUI from here.
         return null;  // return your result
      }
      @Override protected void succeeded(Object result) {
         // Runs on the EDT.  Update the GUI based on
         // the result computed by doInBackground().
      }
   }
```

In der Actionmethode wird nun der entsprechende Task aufgerufen.

Initialisierung, z.B. um Werte aus Textfeldern als Parameter an die Berechnung zu übergeben.

Hier wird die Hintergrundarbeit ausgeführt.

Dies wird nach Beenden des Tasks als Letztes aufgerufen. Hier kann z.B. die GUI wieder aktualisiert werden.

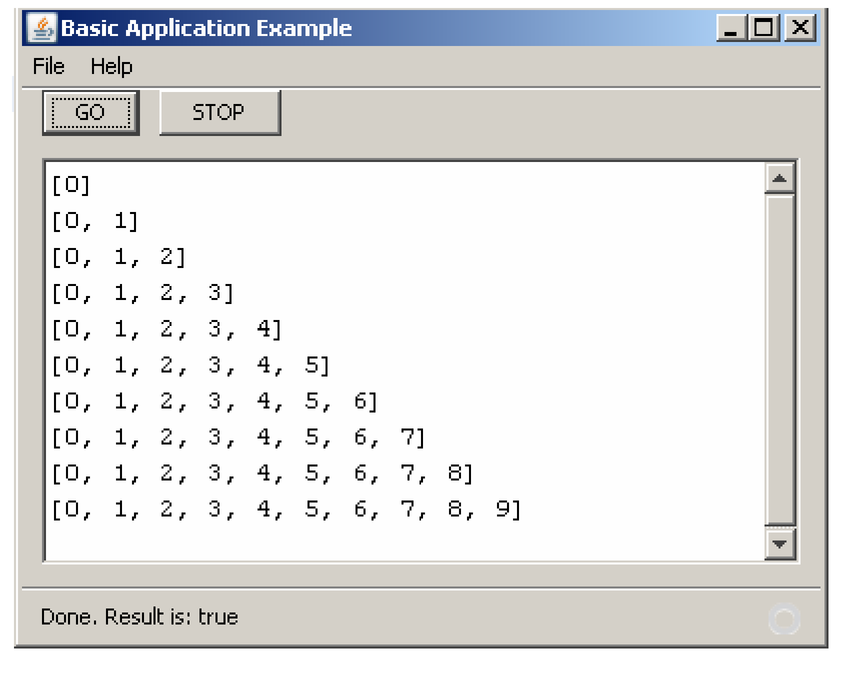
Um zu verstehen, was dieser Code bewirkt ist folgender Artikel sehr hilfreich, da Task den Swingworker einfach erweitert:
http://java.sun.com/developer/technicalArticles/javase/swingworker/

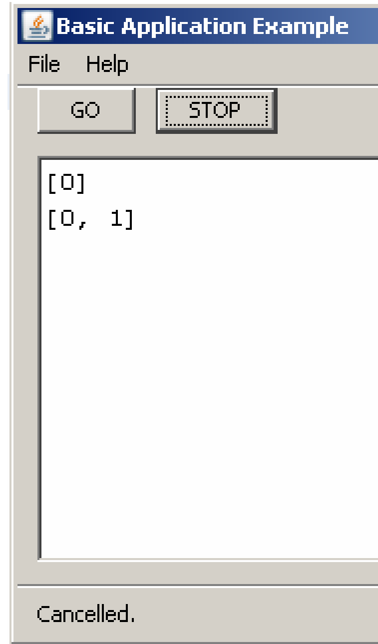
Folgend ein kleines Beispiel, wie dies richtig genutzt werden kann (im Internet finden sich einige fehlerhafte Beispiele, die sehr verwirren können). In dem Beispiel werden 2 Button erzeugt „goTest“ startet den Task. „stopTask“ unterbricht diesen. Zusätzlich gibt es eine TextArea in die Zwischenergebnisse geschrieben werden.

```
231    GoTestTask myTask;
232
233    @Action
234    public Task goTest() {
235       myTask = new GoTestTask(getApplication());          //Hier wird der Task zwischengespeichert
236       return myTask;
237    }
238
239    private class GoTestTask extends org.jdesktop.application.Task<Object, Object> {
                                                              //Achtung z.B. void => Object !
240
241       GoTestTask(org.jdesktop.application.Application app) {
242          super(app);
243          jTextArea1.setText(null);                        //Vor Beginn die Textarea säubern
244       }
245
246       @Override
247       protected Object doInBackground() throws InterruptedException {
```

```
248         ArrayList<Object> ao = new ArrayList<Object>();        //z.B. für Zwischenergebnisse
249         for (int i = 0; i < 10; i++) {
250             if (!isCancelled()) {                              //Wichtig für cancel
251                 setMessage("Working... [" + i + "]");
252                 Thread.sleep(300L);
253                 setProgress(i, 0, 9);                          //Progressbar in Stufen hoch
254             }
255             ao.add(i);
256             publish(ao); // oder auch publish(i);              //Zwischenergebnis veröffentlichen
257         }
258         return true;
259     }
260
261     @Override                                                  //Methode zur Veröffentlichung
262     protected void process(List<Object> values) {              //Achtung hier wieder void=>Object
263         for (Object val : values) {                            //geht auch ohne super.process(values);
264             jTextArea1.append(val.toString()+System.getProperty( "line.separator" ));
265         }
266     }
267
268     @Override
269     protected void succeeded(Object result) {                  //Endergebnis. Wird niemals bei cancel
                                                                   aufgerufen! Falsche Beispiele im Internet.
270         setMessage("Done. Result is: " +result);
271     }
272
273     @Override
274     protected void cancelled() {                               //Wird aufgerufen, wenn cancel
275         super.cancelled();
276         setMessage("Cancelled." );
277     }
278
279   }
280
281   @Action
282   public void stopTask() {
283       myTask.cancel(true);                                     //Task killen
284   }
```

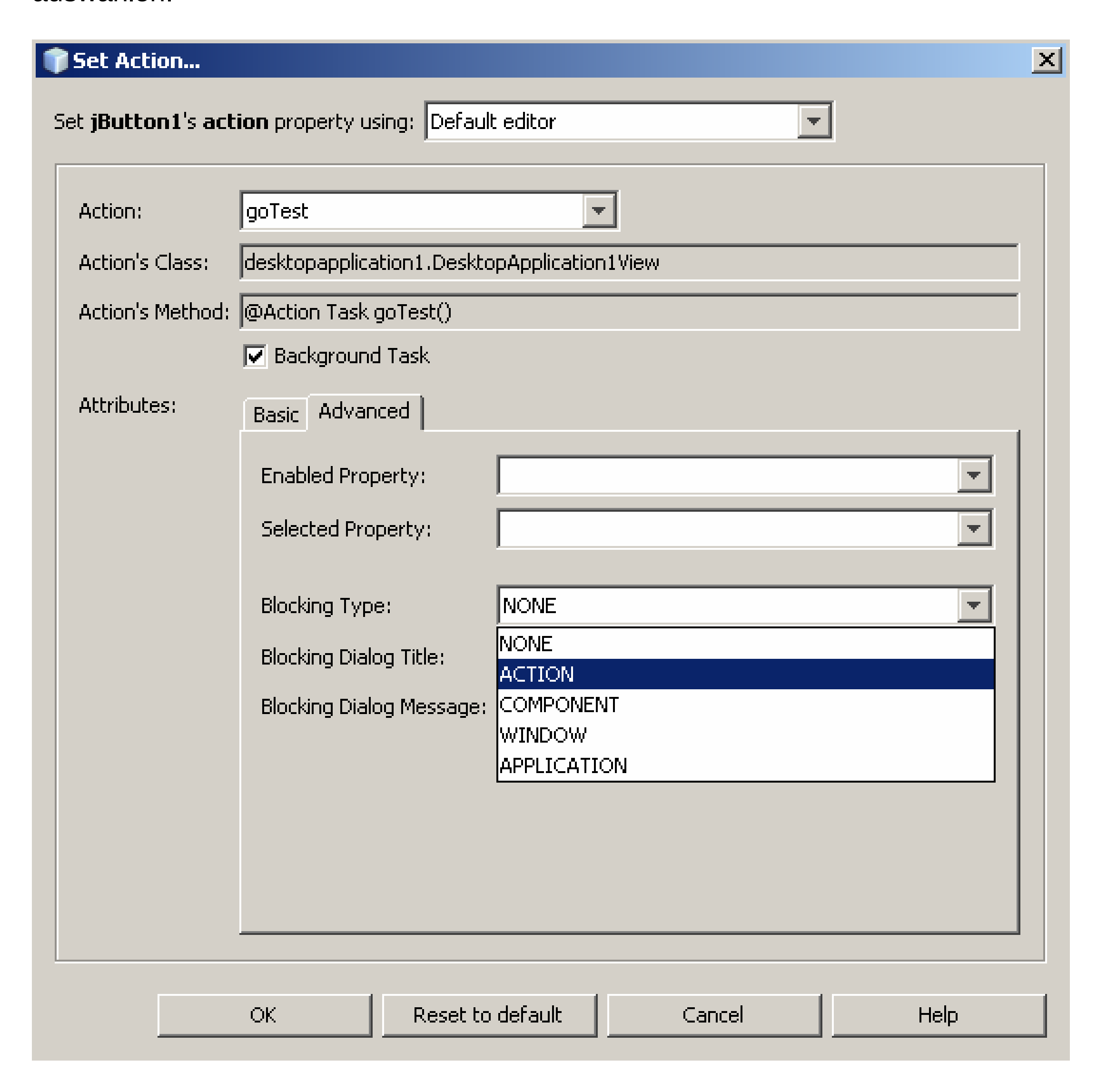
Das Ergebnis sieht dann wie folgt aus:





Ohne Abbruch Mit Stop gedrückt.

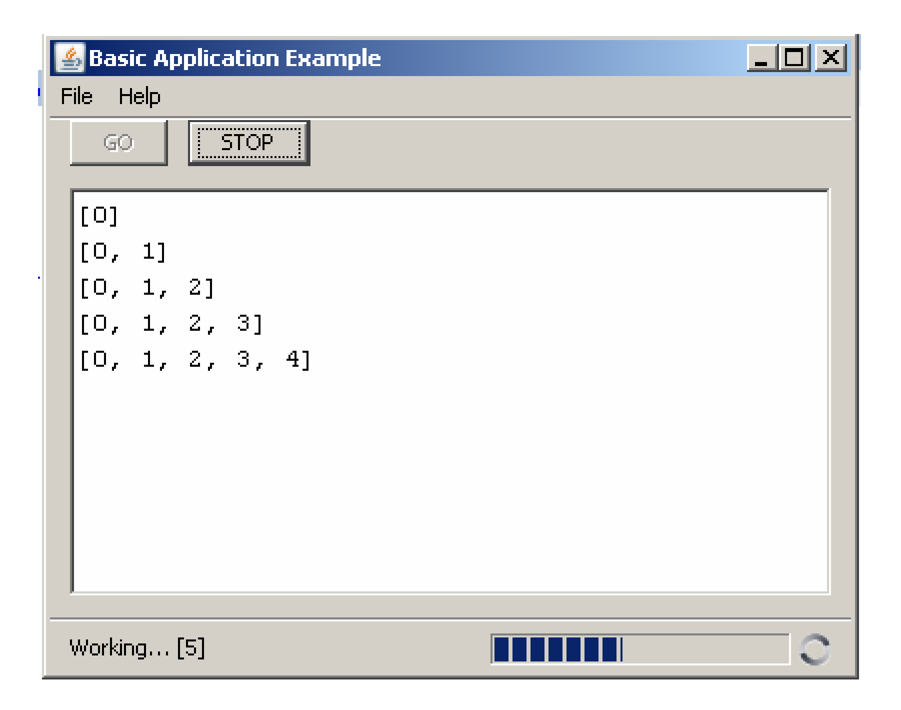
Weiterhin lässt sich sehr einfach einstellen, ob z.B. ein Button nach dem Ausführen geblockt werden soll, solange die Action läuft. Hierzu im ActionDialog in Advanced den Blockingtyp auswählen.



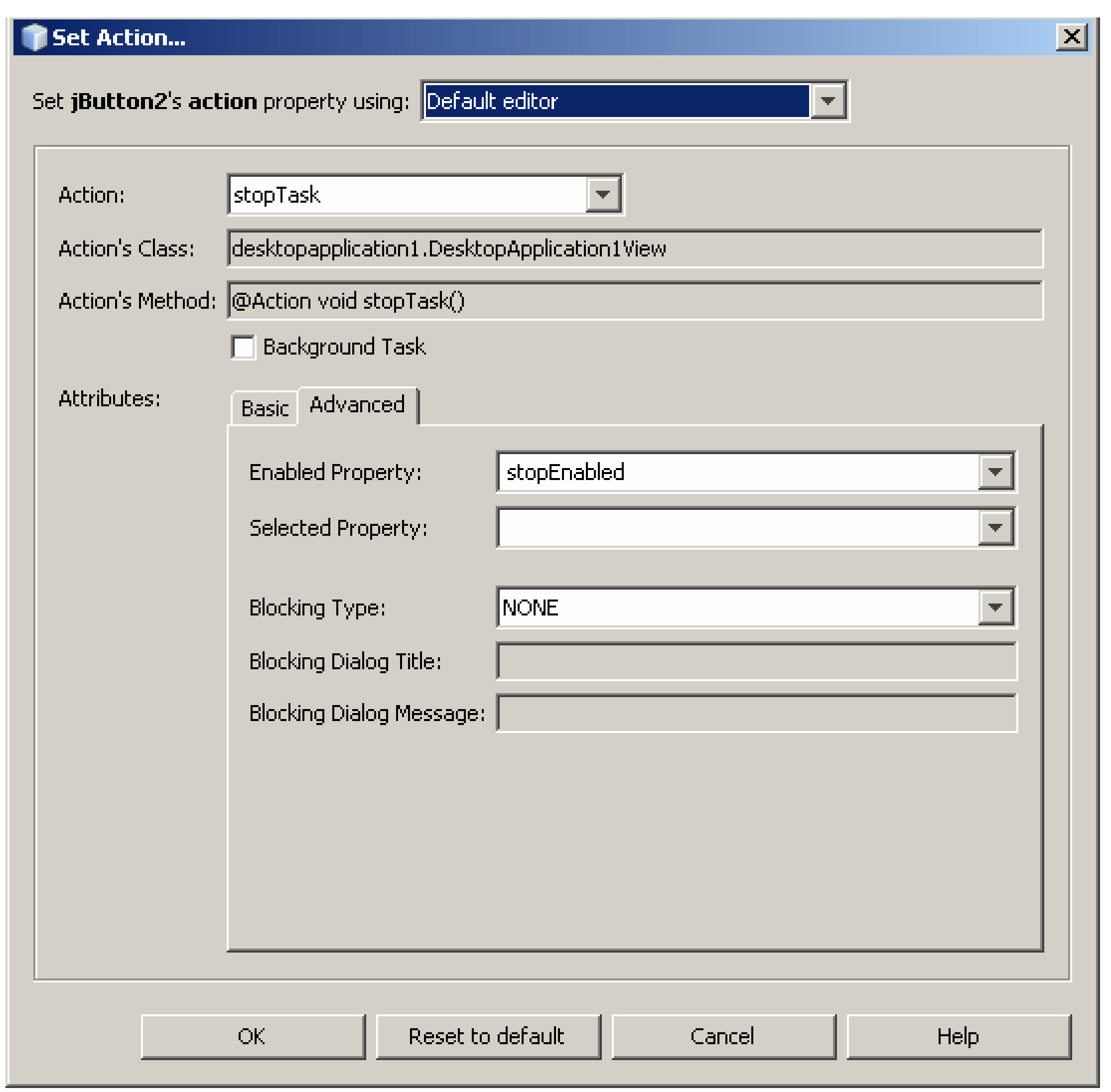
Im Code wird dann die Annotation wie folgt geändert:

```
    @Action(block = Task.BlockingScope.ACTION)
  public Task goTest() {
```

Im Ergebnis sieht dies dann wie folgt aus:



Ein weiteres Feature ist, dass der ActionDialog uns bei der Erstellung von gebundenen Eigenschaften (Stop-Button erst verfügbar, wenn Task läuft) hilft.



Wählen wir unter ‚Advanced' eine Namen für eine Property z.B. „stopEnabled", so erzeugt uns Netbeans folgenden Code:

```
292 private boolean stopEnabled = false;
293 public boolean isStopEnabled() {
294     return stopEnabled;
295 }
296
297 public void setStopEnabled(boolean b) {
298     boolean old = isStopEnabled();
299     this.stopEnabled = b;
300     firePropertyChange("stopEnabled", old, isStopEnabled());
301 }
```

Zusätzlich wird noch die Anotation für die stopAction angepasst:

```
284     @Action(enabledProperty = "stopEnabled")
285 public void stopTask() {
```

Jetzt brauchen wir nur noch beim Aufruf des Tasks und beim Stopp jeweils die Eigenschaft umschalten:

```
236 public Task goTest() {
237     setStopEnabled(true);
238     myTask = new GoTestTask(getApplication());
239     return myTask;
240 }
```

Und:

```
284     @Action(enabledProperty = "stopEnabled")
285 public void stopTask() {
286     myTask.cancel(true);
287     setStopEnabled(false);
288 }
```

15. Desktop Application (Database) - Alexander Zitzer, 20.12.2010

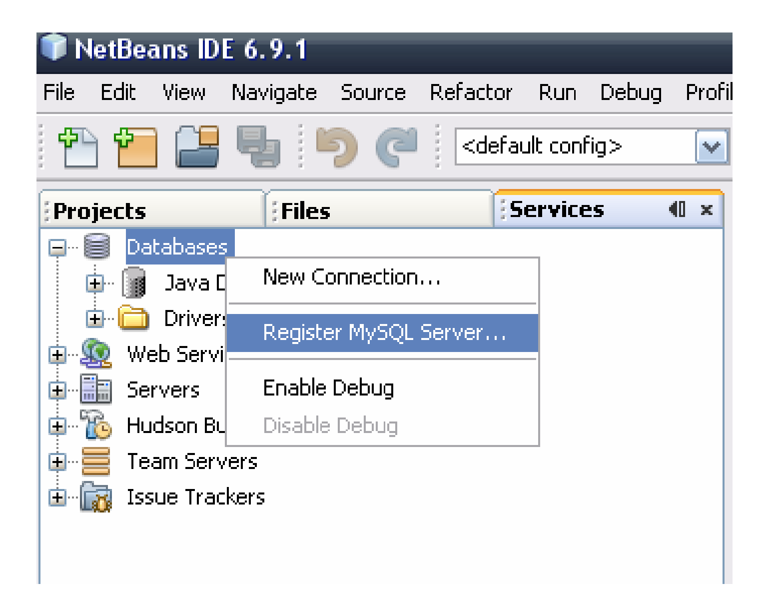
15.1. Datenbanken unter Netbeans

Netbeans bietet uns viele Funktionen um mit Datenbanken zu arbeiten und nimm uns viel Arbeit ab. So können wir in kurzer Zeit Datenbanken erstellen, bearbeiten und in unsere Projekte einbinden.

15.2. MySQL-Datenbank verbinden

Um sich mit MySQL-Datenbank zu verbinden, gehe man wie folgt vor:

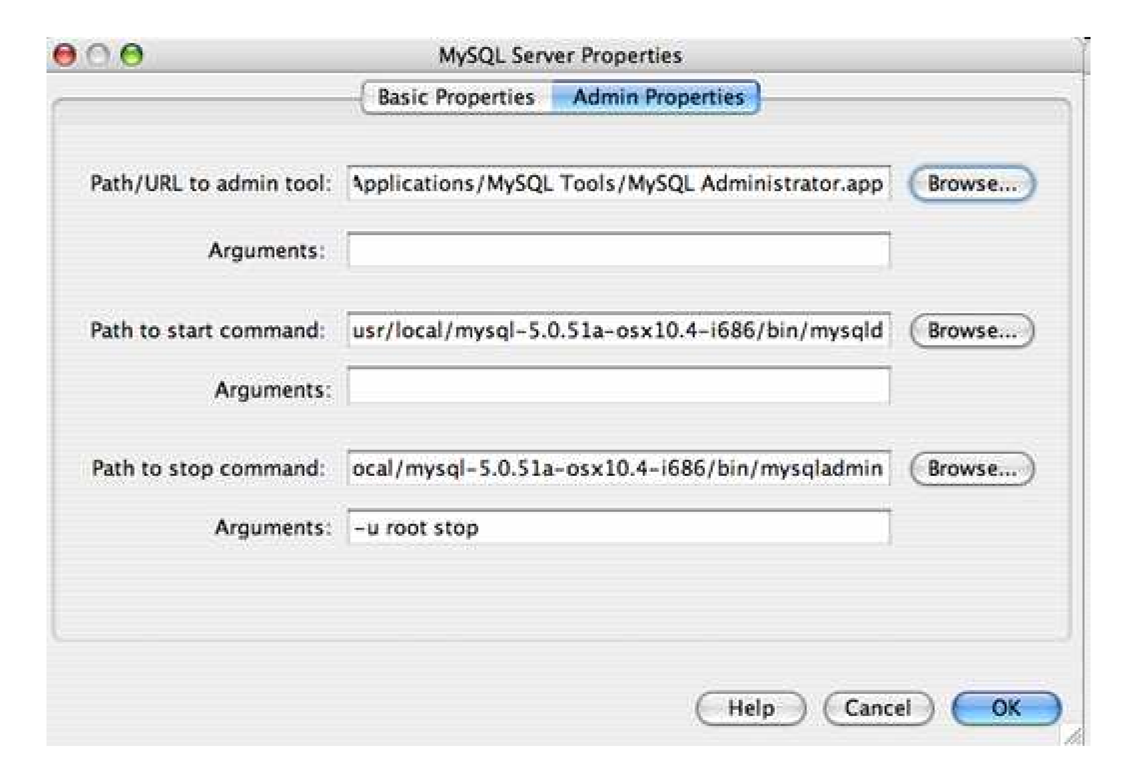
- Netbeans starten und links das Fenster „Services“ aussuchen
- Rechtsklick auf „Databases“ und den Punkt „Register MySQL-Server…“ wählen



- Im folgenden Fenster die Server-Eigenschaften eintragen

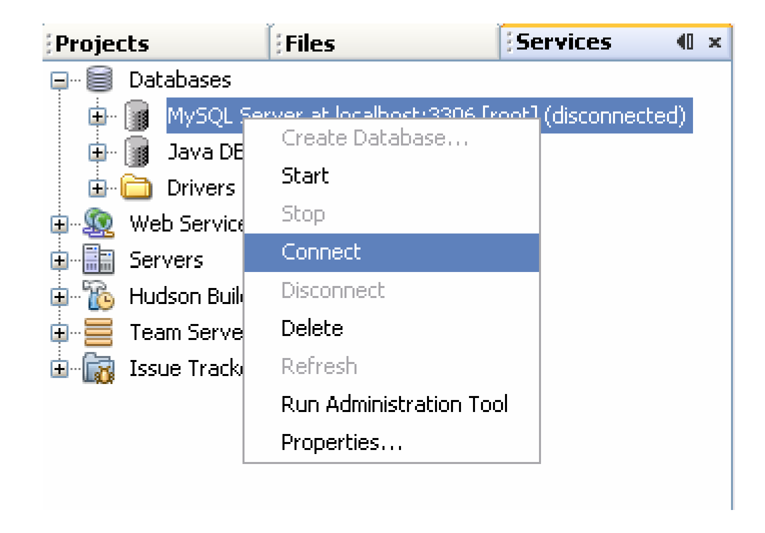


- Zu „Admin Properties“ wechseln und Dateipfade zu angegebenen Dateien angeben



- Path/URL to admin tool:
  C:\Programme\MySQL\MySQL Server 5.5\bin\mysqladmin.exe
- Path to start command:
  C:\Programme\MySQL\MySQL Server 5.5\bin\mysqld.exe
- Path to stop command:
  C:\Programme\MySQL\MySQL Server 5.5\bin\mysqladmin.exe
- Arguments: -u root stopp

Der Server ist jetzt eingerichtet
Falls er noch nicht verbunden ist (disconnected) -> Rechtsklick auf den Server und den Punkt „Connect" wählen



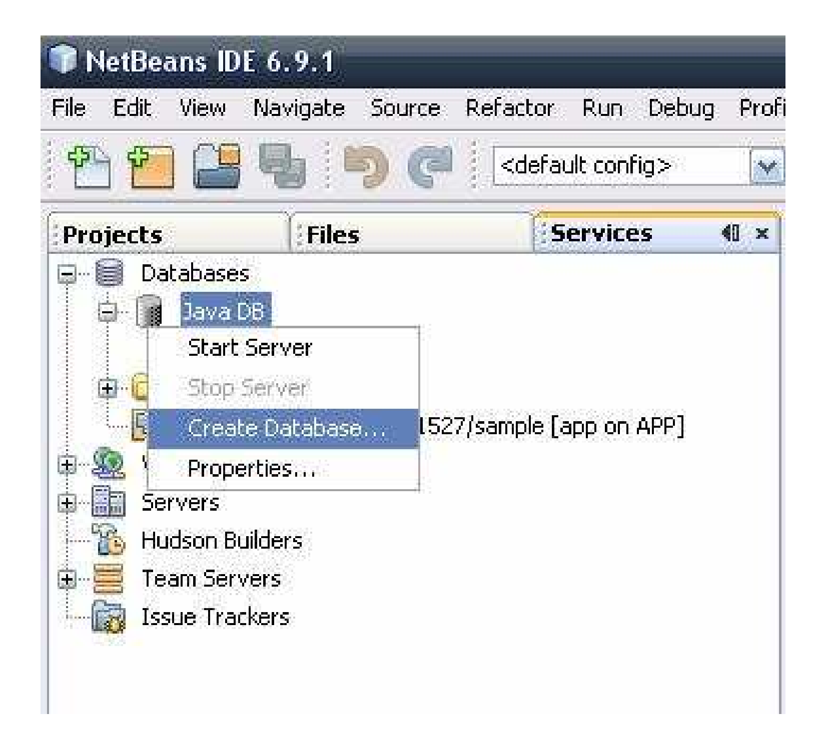
Jetzt ist auch die Verbindung zu dem Server aufgebaut

15.3. Datenbank erstellen und bearbeiten

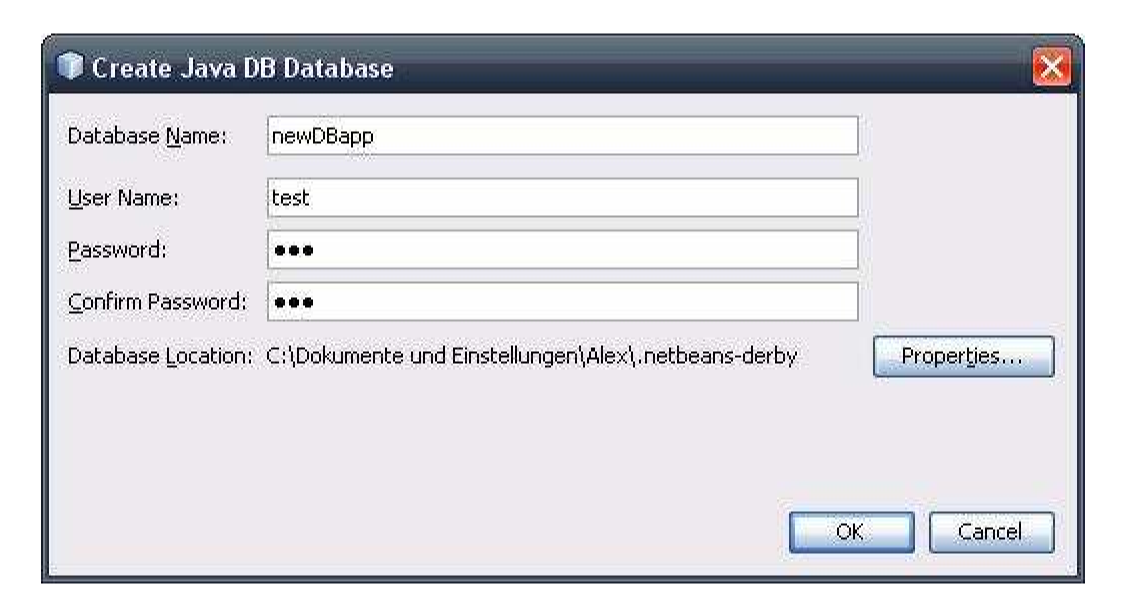
Um eine Datenbank zu erstellen, gehe man wie folgt vor:

- Netbeans starten und links das Fenster „Services" aussuchen

- Rechtsklick auf den Server und den Punkt „Create Database…“ wählen

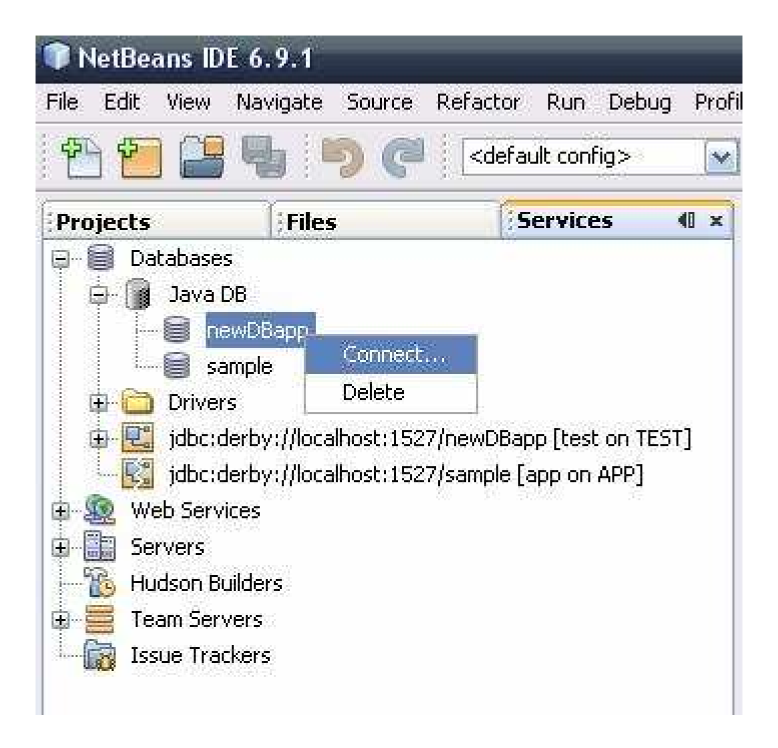


- Die Felder ausfühlen



- Klicke auf „OK“ -> Eine leere Datenbank ist jetzt angelegt

Um die Datenbank zu bearbeiten muss man sich erst mit dieser Datenbank verbinden, indem man auf diese Rechts klickt und den Punkt „Connect…“ auswählt

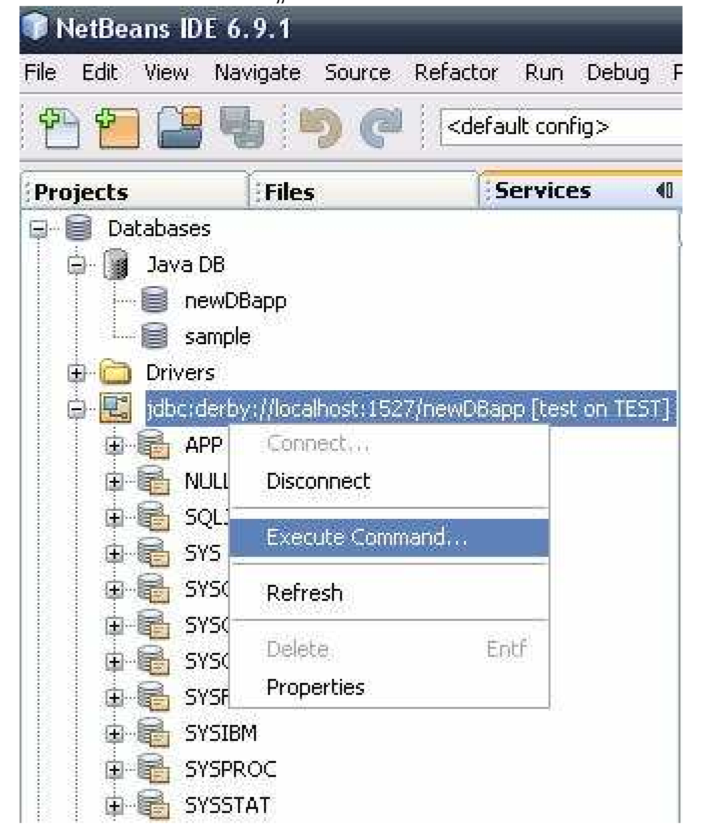


Jetzt ist die Datenbank fertig zum Bearbeiten

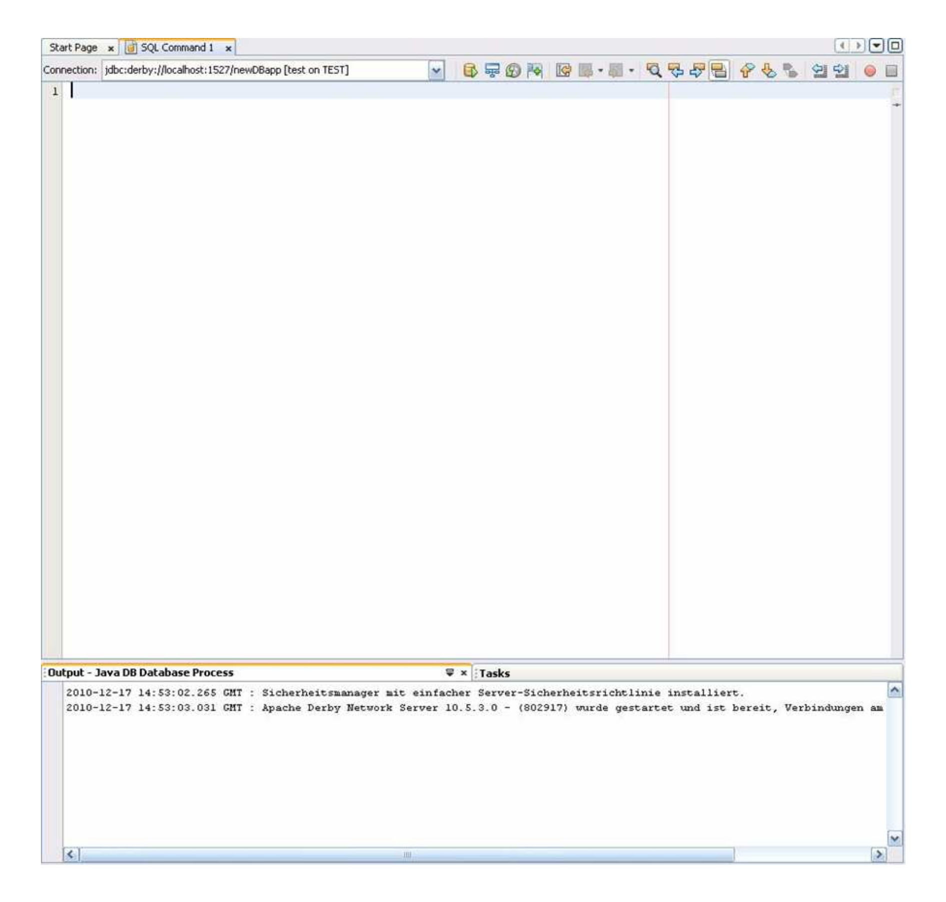
Wir können die Datenbank in zwei verschiedenen Weisen bearbeiten, indem wir die Netbeans-Funktionen nutzen oder indem wir die SQL-Befehle von Hand eintragen.

15.3.1. Datenbank von Hand bearbeiten

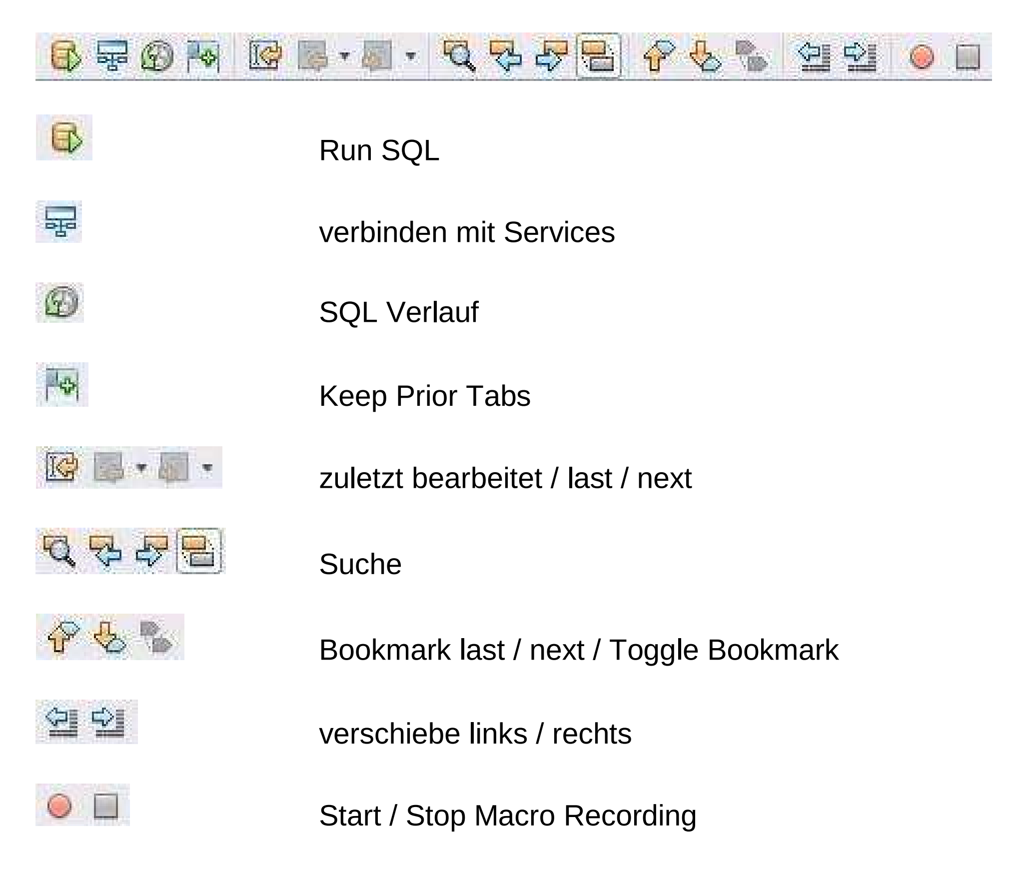
Rechtsklick auf den erstellten Server und „Execute Command…“ wählen



Im folgenden Fenster können wir die SQL-Befehle eingeben und ausführen



Folgende Funktionen bietet Netbeans an:



Run SQL

verbinden mit Services

SQL Verlauf

Keep Prior Tabs

zuletzt bearbeitet / last / next

Suche

Bookmark last / next / Toggle Bookmark

verschiebe links / rechts

Start / Stop Macro Recording

Ein kurzes Script ausgeführt:

```
create table auto (
   id int not null primary key,
   marke varchar(50),
```

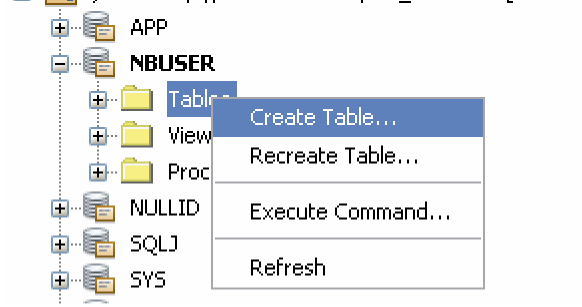
```
        ort varchar(50),
        preis varchar(20)
    );

    create table haendler (
        id int not null primary key,
        h_name varchar(50),
        ort varchar(50),
        email varchar(70)
    );
```

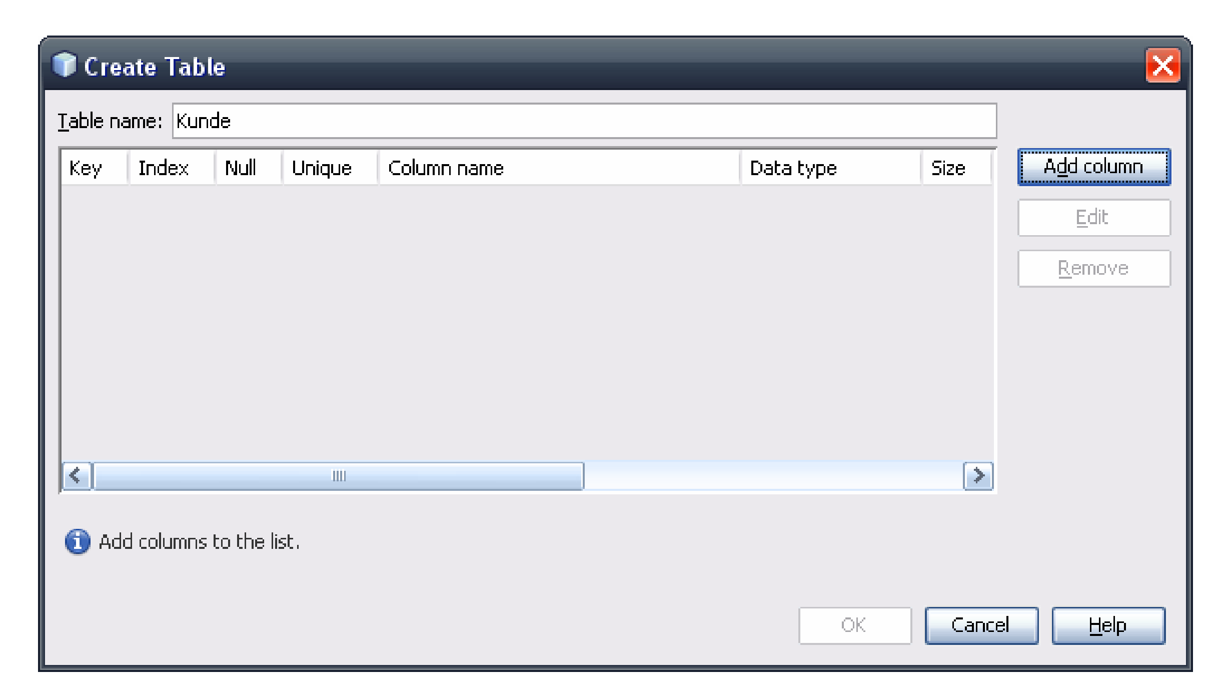
Wir bekommen zwei Tabellen, wie im folgenden Bild zu sehen ist:

15.3.2. Datenbank mit Hilfe von Netbeans bearbeiten

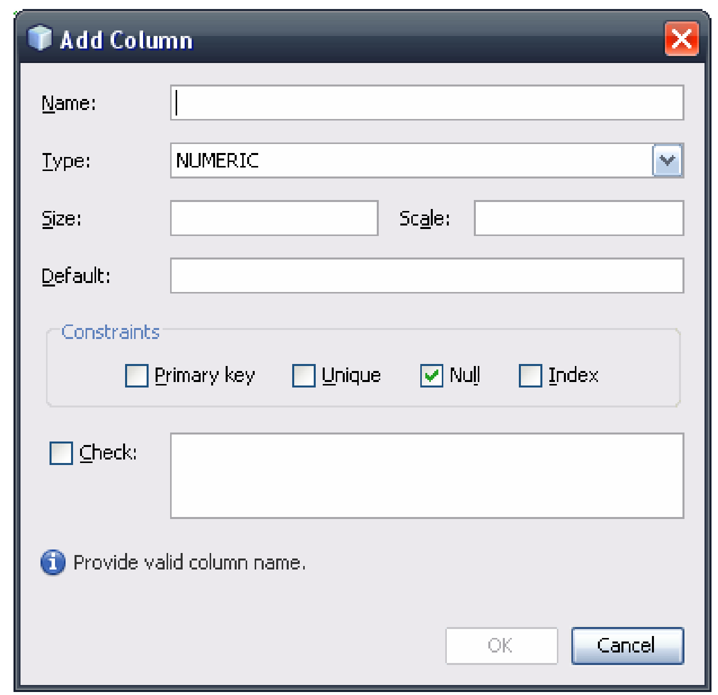
Rechtsklick auf „Tables“ und „Create Table…“ wählen



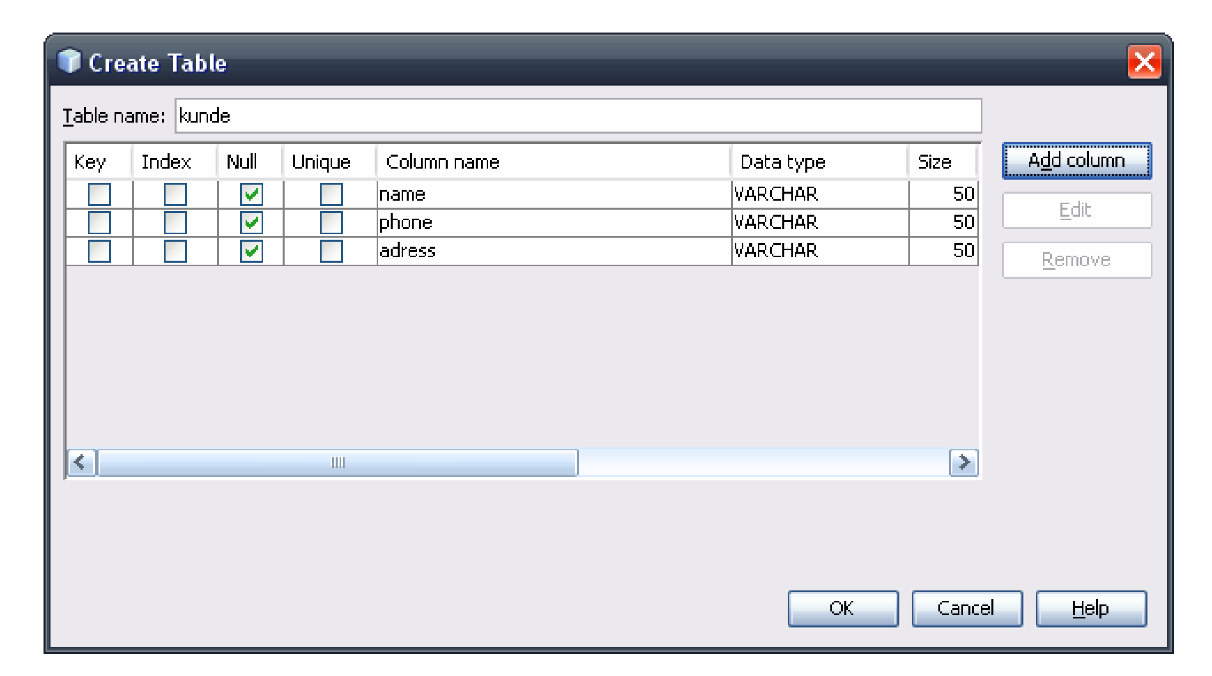
Im folgenden Fenster können wir unsere Tabelle bearbeiten



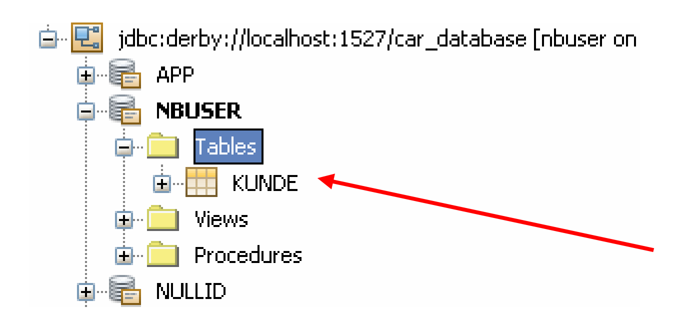
Mit dem Button „Add column“ können wir neue Spalten einfügen



Nach einfügen einiger Spalten bekommen wir folgende Ausgabe:



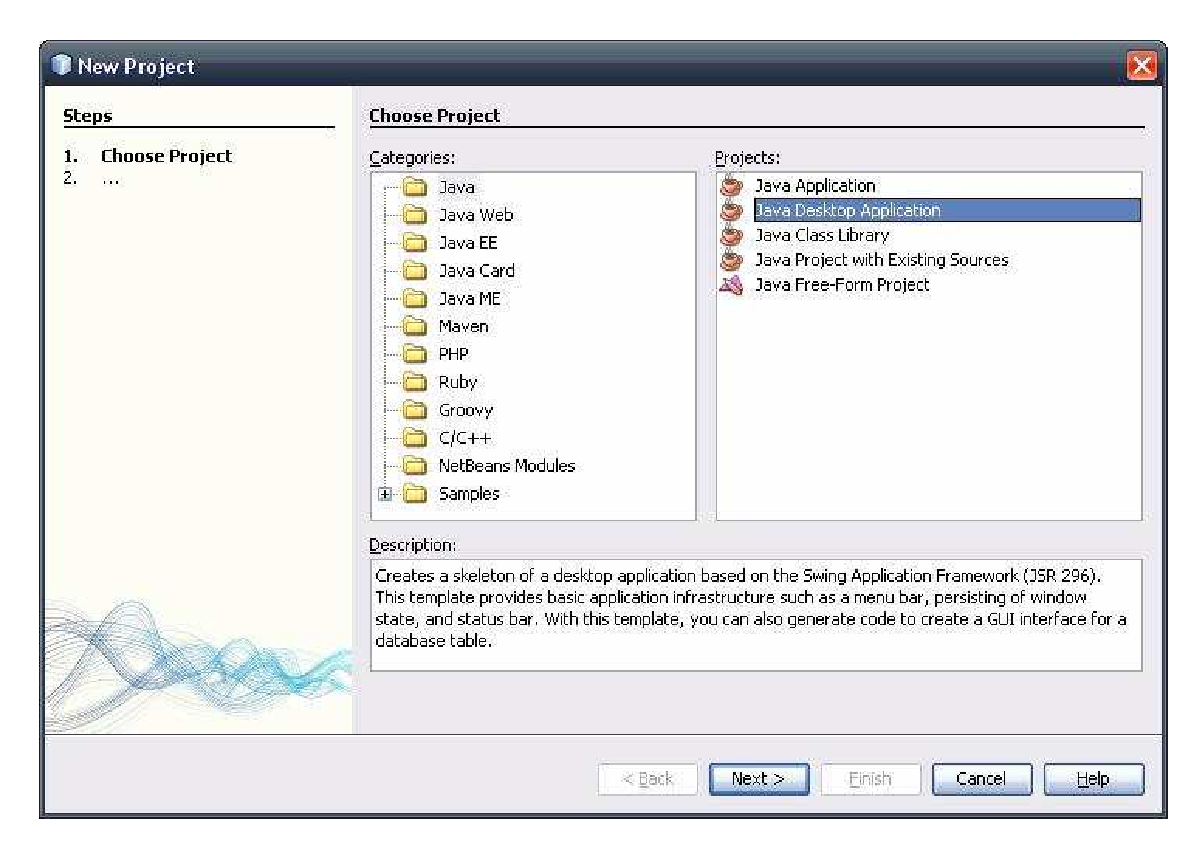
Und hiermit ist unsere Tabelle „kunde“ fertig.



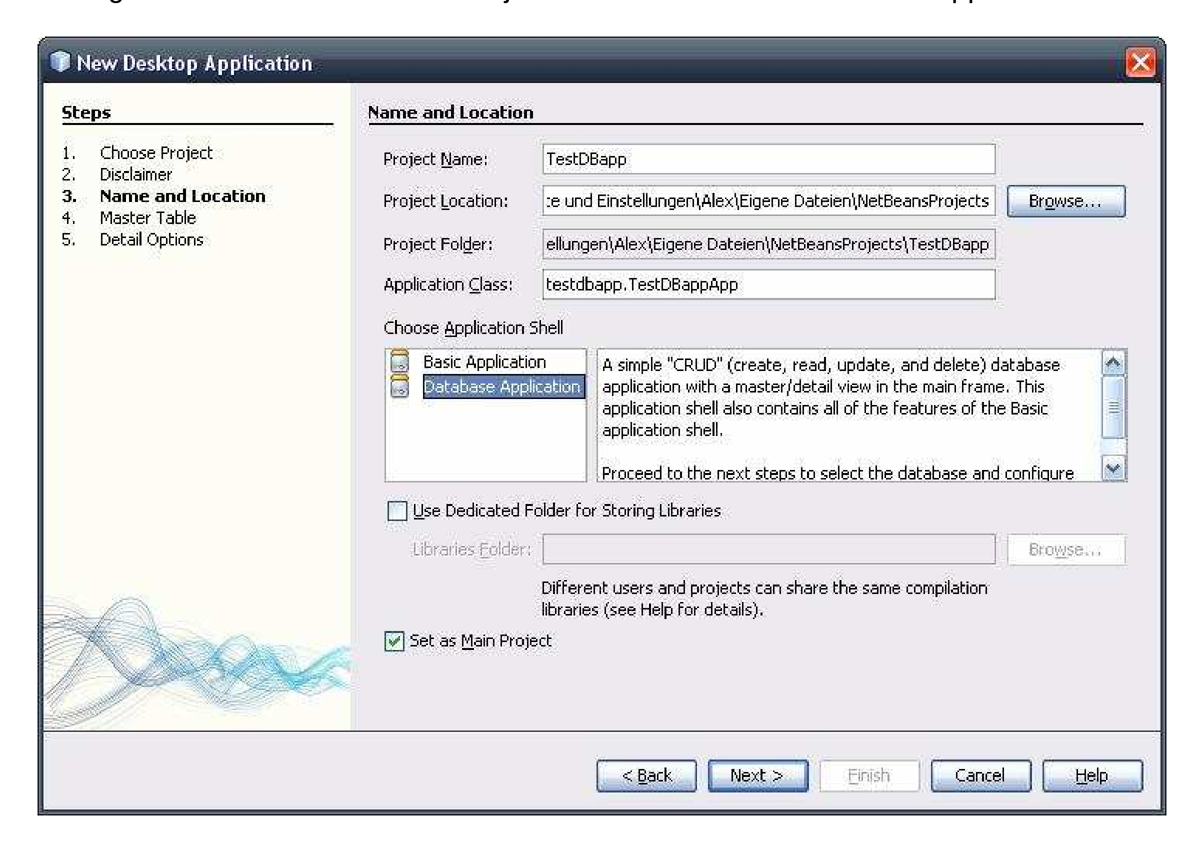
15.3.3. Datenbank einbinden

Um die Datenbank in unser Projekt einzubinden, müssen wir erstmal ein Projekt erstellen und zwar wie folgt:

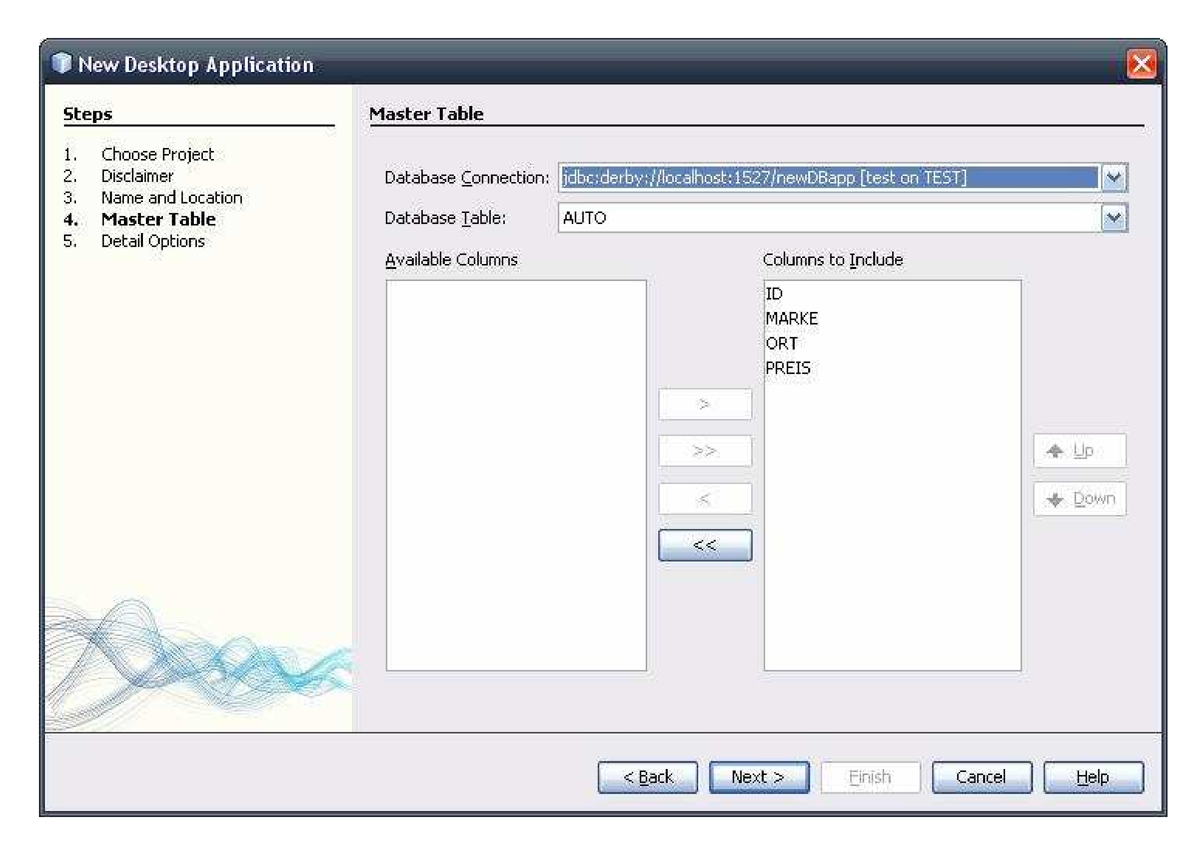
Netbeans starten -> File -> New Project -> Java -> Java Desktop Application



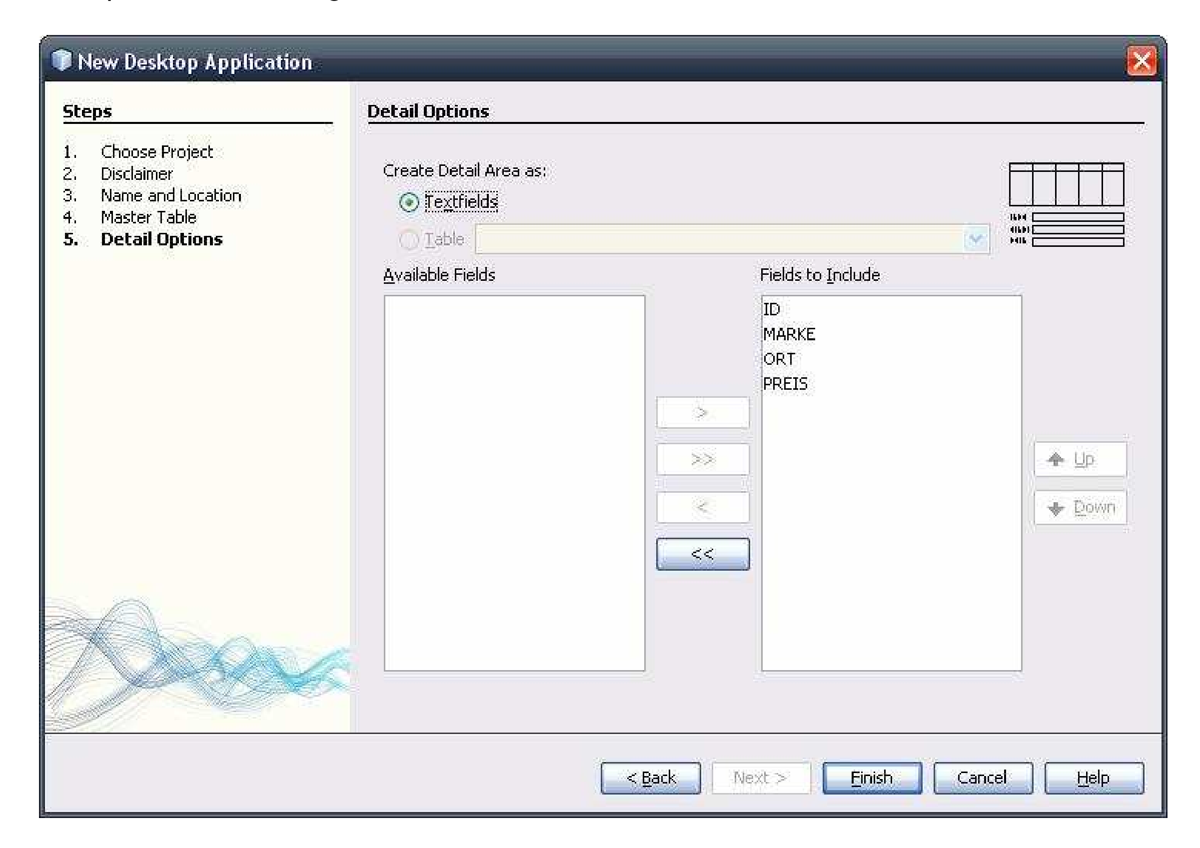
Jetzt geben wir den Namen des Projektes an und wählen “Database Application”



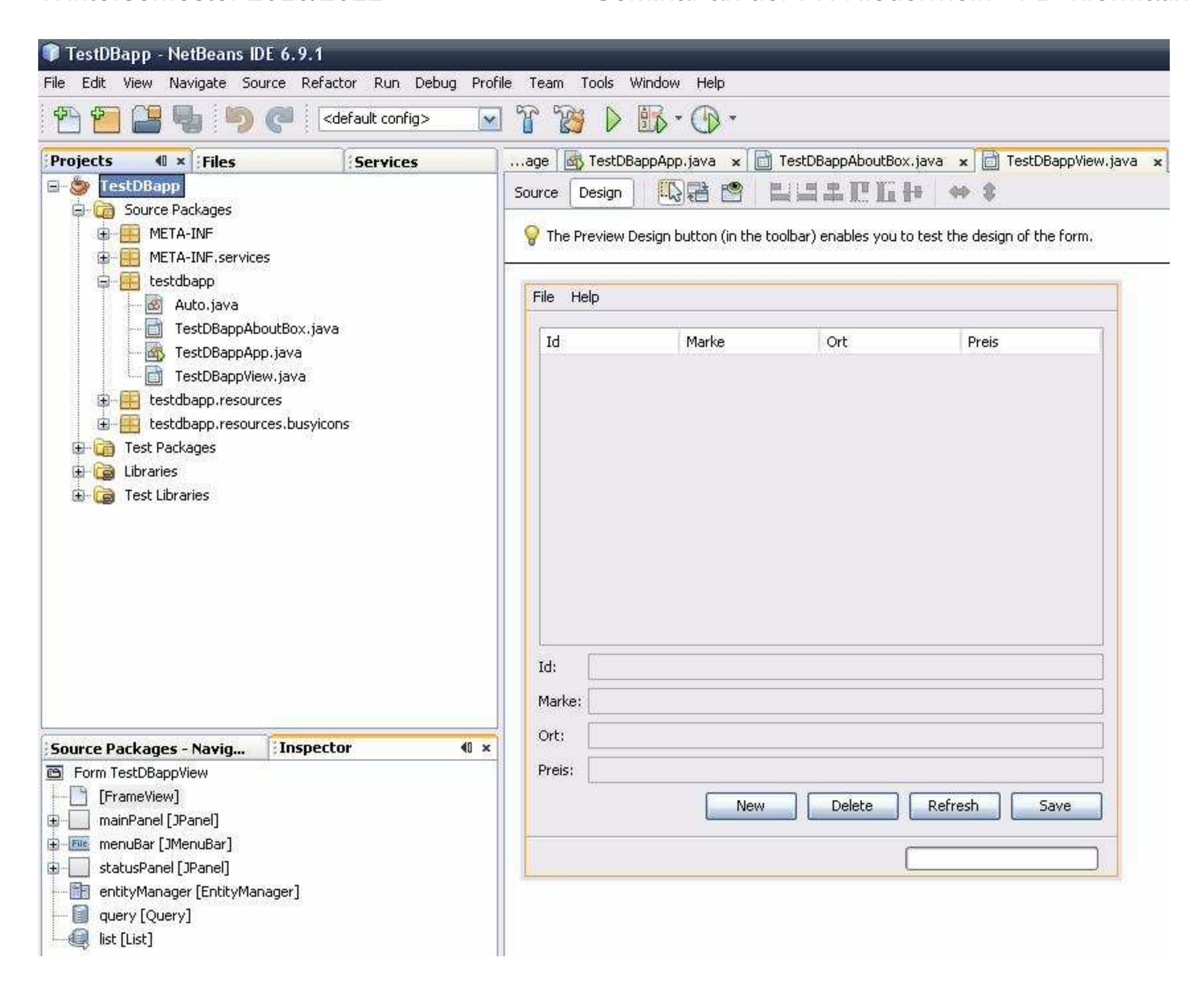
Im folgenden Fenster suchen wir uns die benötigte Datenbank aus und wählen die Tabelle, die wir in unserem Projekt brauchen. Wobei wir einzelne Spalten weglassen können



Jetzt wählen wir das Format der Ausgabe aus (Textfields oder Table) und fügen/entfernen die Spalten in der Ausgabe



Hiermit ist unser Projekt fertig und Netbeans hat den kompletten Quellcode für uns erstellt, den Sie im nächsten Kapitel finden können.



15.4. Code

Analog der Desktop-Application werden wieder die wichtigsten Klassen automatisch von Netbeans erzeugt.

Auto.java

```java
package desktopapplication1;

import ...

@Entity
@Table(name = "AUTO", catalog = "", schema = "TEST")
@NamedQueries({
    @NamedQuery(name = "Auto.findAll", query = "SELECT a FROM Auto a"),
    @NamedQuery(name = "Auto.findById", query = "SELECT a FROM Auto a WHERE a.id = :id"),
    @NamedQuery(name = "Auto.findByMarke", query = "SELECT a FROM Auto a WHERE a.marke = :marke"),
    @NamedQuery(name = "Auto.findByOrt", query = "SELECT a FROM Auto a WHERE a.ort = :ort"),
    @NamedQuery(name = "Auto.findByPreis", query = "SELECT a FROM Auto a WHERE a.preis = :preis")})
public class Auto implements Serializable {
    @Transient
    private PropertyChangeSupport changeSupport = new PropertyChangeSupport(this);
    private static final long serialVersionUID = 1L;
    @Id
    @Basic(optional = false)
    @Column(name = "ID")
    private Integer id;
    @Column(name = "MARKE")
    private String marke;
    @Column(name = "ORT")
    private String ort;
    @Column(name = "PREIS")
    private String preis;

    public Auto() {
    }

    public Auto(Integer id) {
        this.id = id;
    }
    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        Integer oldId = this.id;
        this.id = id;
        changeSupport.firePropertyChange("id", oldId, id);
    }

    public String getMarke() {
        return marke;
    }

    public void setMarke(String marke) {
        String oldMarke = this.marke;
        this.marke = marke;
        changeSupport.firePropertyChange("marke", oldMarke, marke);
    }

    public String getOrt() {
        return ort;
    }

    public void setOrt(String ort) {
        String oldOrt = this.ort;
        this.ort = ort;
        changeSupport.firePropertyChange("ort", oldOrt, ort);
    }

    public String getPreis() {
        return preis;
    }
```

```
    public void setPreis(String preis) {
        String oldPreis = this.preis;
        this.preis = preis;
        changeSupport.firePropertyChange("preis", oldPreis, preis);
    }

    @Override
    public int hashCode() {
        int hash = 0;
        hash += (id != null ? id.hashCode() : 0);
        return hash;
    }

    @Override
    public boolean equals(Object object) {
        if (!(object instanceof Auto)) {
            return false;
        }
        Auto other = (Auto) object;
        if ((this.id == null && other.id != null) || (this.id != null && !this.id.equals(other.id))) {
            return false;
        }
        return true;
    }

    @Override
    public String toString() {
        return "desktopapplication1.Auto[id=" + id + "]";
    }

    public void addPropertyChangeListener(PropertyChangeListener listener) {
        changeSupport.addPropertyChangeListener(listener);
    }

    public void removePropertyChangeListener(PropertyChangeListener listener) {
        changeSupport.removePropertyChangeListener(listener);
    }

}
```

DesktopApplication1 (Quellcode siehe oben, da völlig analog)

DesktopApplication1View

```
package desktopapplication1;

import ...

class DesktopApplication1View extends FrameView {

    public DesktopApplication1View(SingleFrameApplication app) {
        super(app);

        initComponents();

        ResourceMap resourceMap = getResourceMap();
        int messageTimeout = resourceMap.getInteger("StatusBar.messageTimeout");
        messageTimer = new Timer(messageTimeout, new ActionListener() {
            public void actionPerformed(ActionEvent e) {...}
        });
        messageTimer.setRepeats(false);
        int busyAnimationRate = resourceMap.getInteger("StatusBar.busyAnimationRate");
        for (int i = 0; i < busyIcons.length; i++) {
            busyIcons[i] = resourceMap.getIcon("StatusBar.busyIcons[" + i + "]");
        }
        busyIconTimer = new Timer(busyAnimationRate, new ActionListener() {
            public void actionPerformed(ActionEvent e) {...}
        });
        idleIcon = resourceMap.getIcon("StatusBar.idleIcon");
        statusAnimationLabel.setIcon(idleIcon);
        progressBar.setVisible(false);

        TaskMonitor taskMonitor = new TaskMonitor(getApplication().getContext());
        taskMonitor.addPropertyChangeListener(new java.beans.PropertyChangeListener() {
            public void propertyChange(java.beans.PropertyChangeEvent evt) {...}
        });

        masterTable.getSelectionModel().addListSelectionListener(
            new ListSelectionListener() {
                public void valueChanged(ListSelectionEvent e) {...}
            });

        bindingGroup.addBindingListener(new AbstractBindingListener() {
            @Override
            public void targetChanged(Binding binding, PropertyStateEvent event) {...}
        });
        entityManager.getTransaction().begin();
    }


    public boolean isSaveNeeded() {...}

    private void setSaveNeeded(boolean saveNeeded) {...}

    public boolean isRecordSelected() {...}
```

```java
    @Action
    public void newRecord() {
        desktopapplication1.Auto a = new desktopapplication1.Auto();
        entityManager.persist(a);
        list.add(a);
        int row = list.size()-1;
        masterTable.setRowSelectionInterval(row, row);
        masterTable.scrollRectToVisible(masterTable.getCellRect(row, 0, true));
        setSaveNeeded(true);
    }

    @Action(enabledProperty = "recordSelected")
    public void deleteRecord() {
        int[] selected = masterTable.getSelectedRows();
        List<desktopapplication1.Auto> toRemove = new ArrayList<desktopapplication1.Auto>(selected.length);
        for (int idx=0; idx<selected.length; idx++) {
            desktopapplication1.Auto a = list.get(masterTable.convertRowIndexToModel(selected[idx]));
            toRemove.add(a);
            entityManager.remove(a);
        }
        list.removeAll(toRemove);
        setSaveNeeded(true);
    }


    @Action(enabledProperty = "saveNeeded")
    public Task save() {...}
    private class SaveTask extends Task {
        SaveTask(org.jdesktop.application.Application app) {...}
        @Override protected Void doInBackground() {...}
        @Override protected void finished() {...}
    }

    @Action
    public Task refresh() {...}
    private class RefreshTask extends Task {
        RefreshTask(org.jdesktop.application.Application app) {...}
        @SuppressWarnings("unchecked")
        @Override protected Void doInBackground() {...}
        @Override protected void finished() {...}
    }
    @Action
    public void showAboutBox() {...}
    @SuppressWarnings("unchecked")
    Generated Code
    variables declaration

    private final Timer messageTimer;
    private final Timer busyIconTimer;
    private final Icon idleIcon;
    private final Icon[] busyIcons = new Icon[15];
    private int busyIconIndex = 0;

    private JDialog aboutBox;

    private boolean saveNeeded;
}
```

save() und refresh() werden als Backgroundtask erzeugt

16. Internationalisierung – Paul Bernasch, 20.12.2010

16.1. Was ist Internationalisierung?

Internationalisierung ist ein Tool, das das Anpassen eines Programms an eine andere Sprache, als die Sprache in der es anfänglich programmiert wurde, erleichtert.

16.1.1. Welche Möglichkeiten stehen uns durch die Internationalisierung offen?

Wir können nun mit wenigen Klicks Strings anpassen, das Programm somit in eine andere Sprache übersetzen, aber wir können auch die Darstellung des Datums, die Maßeinheit für die Längen und die Darstellung der Währung ändern.

16.1.2. Der Begriff I18N

I18N ist eine Abkürzung für den Begriff Internationalization, da man mit dem ausschreiben des Begriffes lange beschäftigt ist wurde wahrscheinlich diese Abkürzung eingeführt.

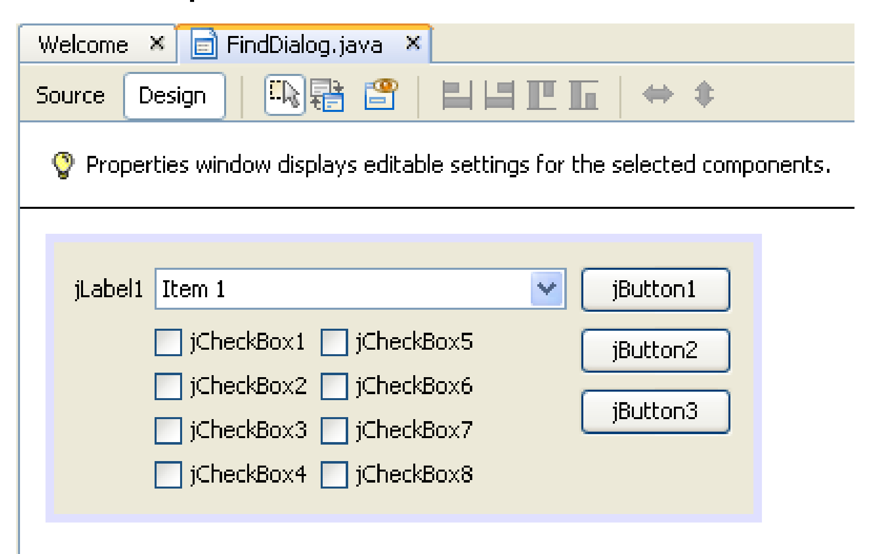
Bei I18N steht das I für den den ersten Buchstaben in Internationalization, die 18 soll die 18 folgenden Buchstaben ersetzen und das N ist das letzte N und somit der letzte Buchstabe des Wortes Internationalization.

16.2. Wie funktioniert Internationalisierung?

16.2.1. Internationalisierung einer einzelnen Datei

Die veränderbaren Strings werden in einer .Properties-Datei als Key/Value Paare gespeichert. Somit ist es einfacher die Programme zu internationalisieren, da nur die Values angepasst werden müssen und nicht der ganze Code umgeschrieben werden muss.

Hier ein **Beispiel** anhand einer vorher erstellten Benutzeroberfläche.

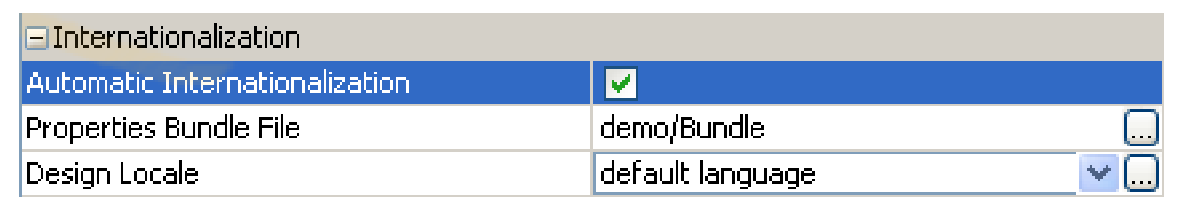


Hier sieht man die Benutzeroberfläche wie sie im Allgemeinen aussieht, nachdem man einige Elemente einem Dialog hinzugefügt hat.

Als erster Schritt der Internationalisierung weißt man den Elementen, Werte (Bezeichnungen) zu, die in einer externen Datei, der **Properties-datei [** die durch Markieren der Automatic Internationalization Option in den Eigenschaften der Form-Datei erstellt wird **]**,

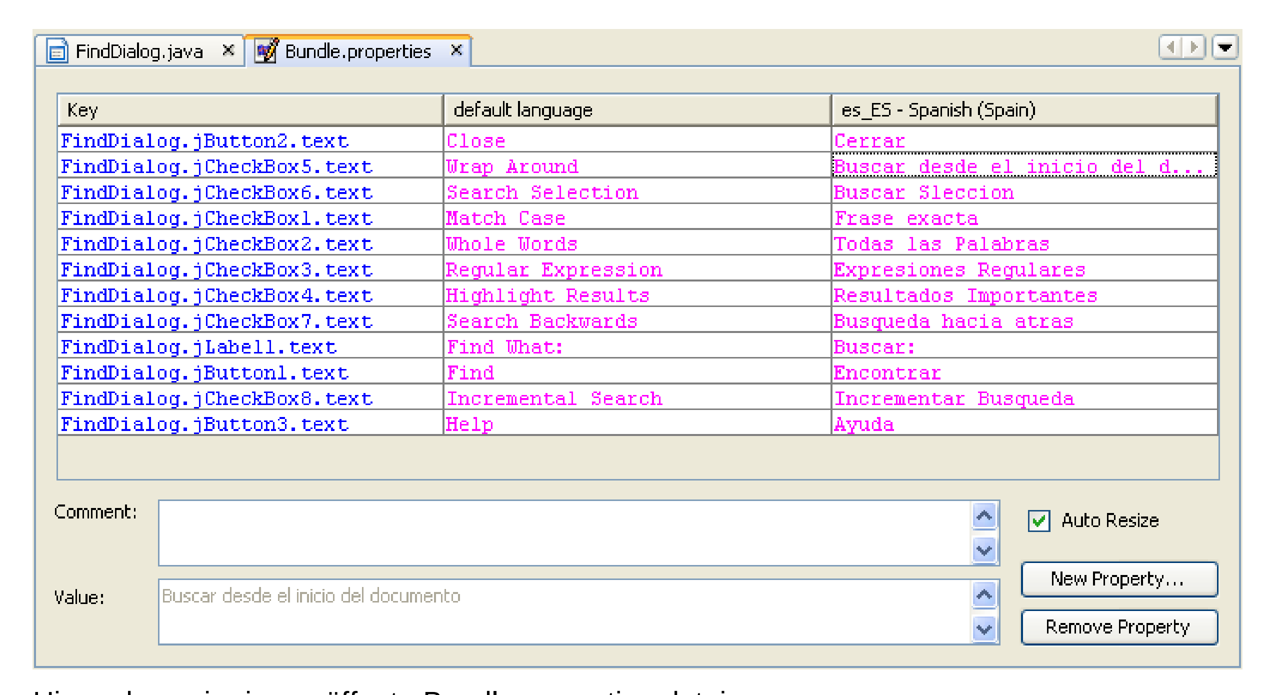
abgespeichert werden. Nun hat man die Möglichkeit beliebig viele Sprachen hinzu zufügen, in dem man eine **Locale-properties-datei** erstellt **[**Rechtsklick auf dem jeweiligen Package dann New-> Properties-File**]**.

Für GUIs lässt sich die Internationalisierung automatisch im Inspector-Fenster aktivieren:



In der Locale-datei werden die jeweilige Strings in einer anderen Sprache abgespeichert - vorteilhaft hier bei ist auch die passende Benennung der Locale nach dem Schema: <Dateiname><Sprache><Land> zu speichern z.B. Editor_de_DE.properties

Das Zuweisen der Werte erfolgt nun durch bearbeiten der ersten Properties-datei, diese speichert dann die neuzugewiesen Werte in der jeweiligen Locale-Properties-Datei ab dies sieht dann wie folgt aus :



Key	default language	es_ES - Spanish (Spain)
FindDialog.jButton2.text	Close	Cerrar
FindDialog.jCheckBox5.text	Wrap Around	Buscar desde el inicio del d...
FindDialog.jCheckBox6.text	Search Selection	Buscar Seleccion
FindDialog.jCheckBox1.text	Match Case	Frase exacta
FindDialog.jCheckBox2.text	Whole Words	Todas las Palabras
FindDialog.jCheckBox3.text	Regular Expression	Expresiones Regulares
FindDialog.jCheckBox4.text	Highlight Results	Resultados Importantes
FindDialog.jCheckBox7.text	Search Backwards	Busqueda hacia atras
FindDialog.jLabel1.text	Find What:	Buscar:
FindDialog.jButton1.text	Find	Encontrar
FindDialog.jCheckBox8.text	Incremental Search	Incrementar Busqueda
FindDialog.jButton3.text	Help	Ayuda

Hier sehen wir eine geöffnete Bundle.properties-datei.

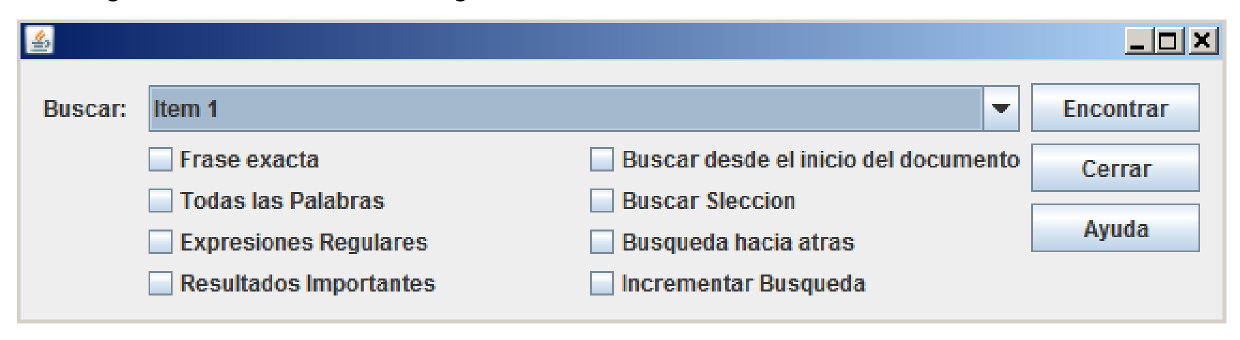
In der linken Spalte haben wir die Keys, die die Bezeichnung aus der oben gezeigten Benutzeroberfläche tragen. Die mittlere Spalte hier enthält englische Begriffe, das sind die zu den Keys gehörenden Values aus einer Englischen Locale-datei, die als default language also Standardsprache gewählt wurde.

In der rechten Spalte sehen wir nun die Werte, die in einer anderen Locale, hier einer spanischen, abgespeichert sind. Die Werte können durch ein Klicken in die jeweilige Zelle bearbeitet werden.

Wenn man nun eine weitere Sprache hinzufügen möchte, so ist der einfachste Weg im Inspector unter Design Locale eine neue SPrahce auszuwählen. Netbeans legt dann automatisch die passende Bundle-datei an.

Die angelegt Locale-Properties-datei lässt sich nun über die Bundle.properties.datei bearbeiten, in unserem Beispiel-bild müsste man dazu nach rechts scrollen um den jeweiligen Zellen wieder die passenden Values zuweisen zu können.

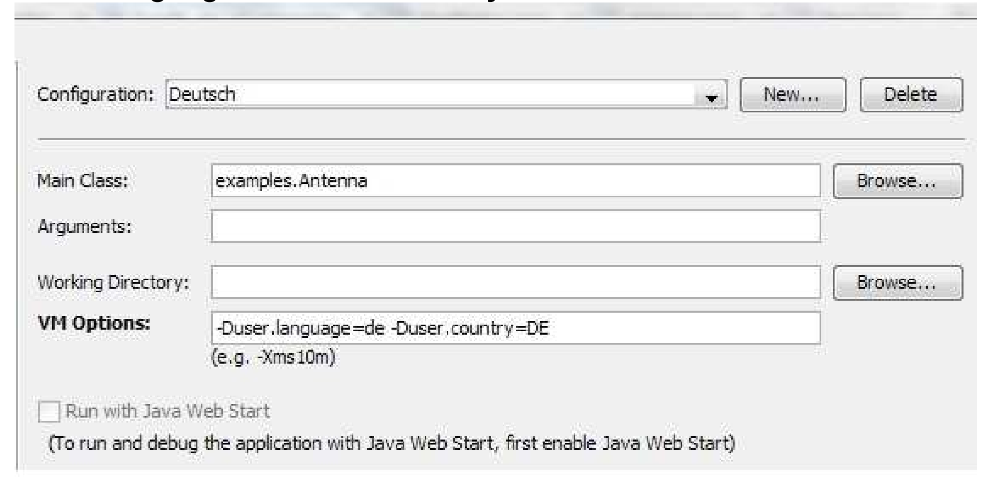
Das Ergebnis könnte dann wie folgt aussehen:



Um die einzelnen Sprachen einstellen zu können, gibt es zwei Wege. Einer ist über den Code hier muss man sich der **getBundle()-Methode** bedienen. Diese ruft man mit dem Befehl <Klassenname>.getbundle() auf. Als Argumente ruft man dann die Resource-datei auf und die gewünschte Locale. Dieser Befehl könnte dann z.B

```
ResourceBundle.getBundle("MyResources", currentLocale);
```

sein.
Eine weitere Möglichkeit die Sprache festzulegen, ist das Voreinstellen der gewünschten Locale-Datei über die VM.
Dies läuft wie folgt ab. Im Hauptfenster gibt es die Option Customize.
Hier kann man dann der Einstellung einen Namen geben zum Beispiel „Deutsch“ und schreibt dann in die Zeile VM-Options den Befehl: **-Duser.language=de -Duser.country=DE.**
Das Variable an diesem Befehl sind die Kürzel. Das erste Kürzel gibt die Sprache an, in diesem Fall de für Deutsch und das zweite Kürzel gibt das Land an in diesem Fall de für Deutschland.
Ein Unterschied in dem Befehl gibt es bei Amerikanischem Britisch hier würde der Befehl **-Duser.language=en -Duser.country=US** heißen.



Hier sehen wir das Fenster in das wir den Befehl eingeben müssen .

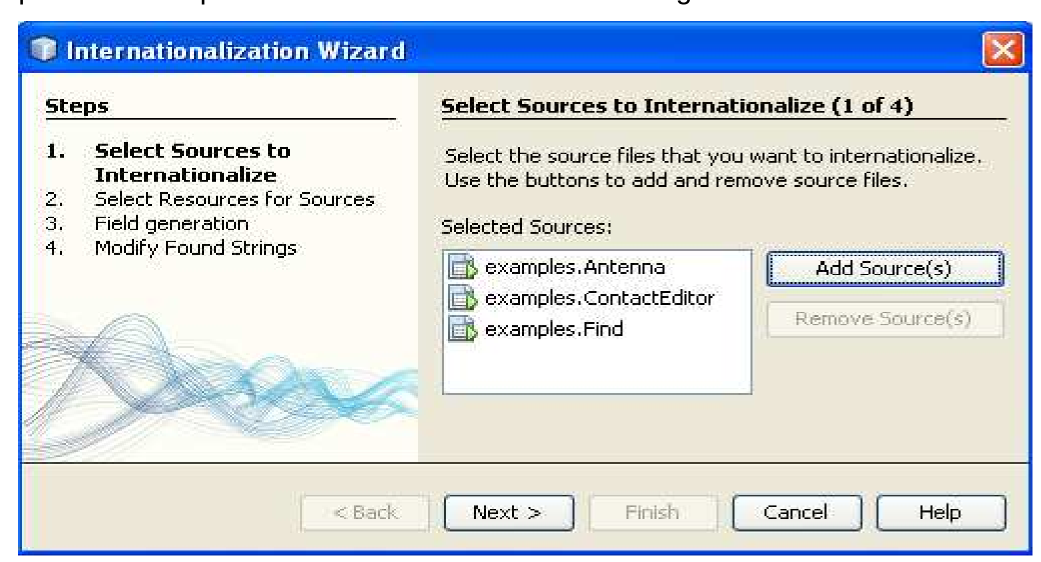
16.2.2. Internationalisieren eines gesamten Projektes

Im Prinzip läuft die Internationalisierung eines ganzen Projektes ab wie bei der Internationalisierung der einzelnen Datei, nur das es sich hier anbietet den

Internationalisierungs-Wizard zu benutzen. Dieser ist im Netbeans-Hauptfenster über Tools -> Internationalization -> Internationalization Wizard zu erreichen.

Als erster Schritt muss man jedoch wieder Properties-dateien, wie vorher beschrieben, erstellen, damit der Internationalization Wizard einen Speicherort hat.

Als nächstes starten wir den Internationalization Wizard. Der Internationalization Wizard geht in vier Schritten vor, als erstes sucht man sich eine Quelldatei und fügt dieser dann die passende Properties hinzu. Dies sieht dann wie folgt aus:



1.Schritt

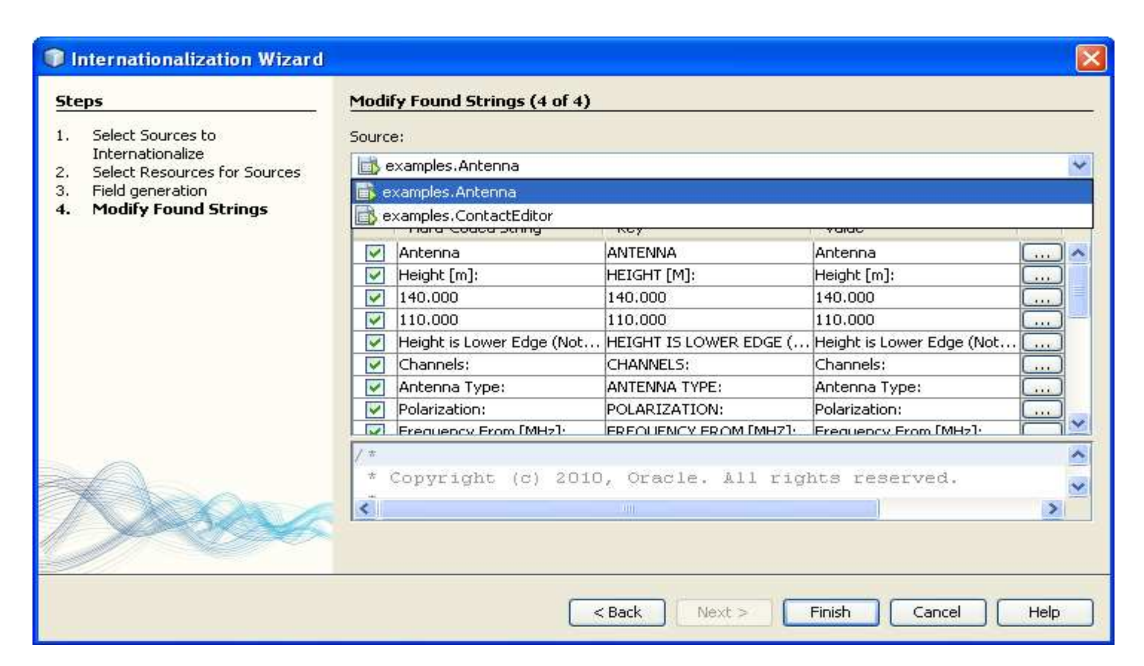


2. Schritt

Im dritten Schritt des Wizards kann man noch der Quelldatei noch weitere Eigenschaften hinzufügen um einfacher auf die Datei zugreifen zu können.

Im nächsten Schritt kann man nun alle Strings aus der Quelldatei bearbeiten und nun werden die Strings der Dateien, die die gleiche Properties-Datei verwenden, direkt angepasst.

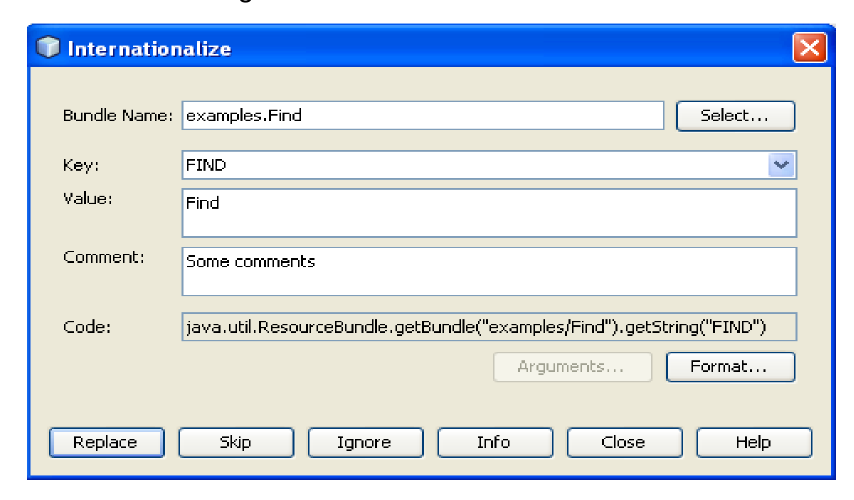
Der vierte Schritt des Wizards sieht somit wie folgt aus:



Links sind die Hard-Coded-Strings, in der Mitte die Keys und rechts die Values.

16.2.3. Internationalisierung einzelner Formen

Wie wir bis jetzt ganze Dateien oder auch ganze Projekte editiert haben, können wir natürlich auch einzelne Strings/ Formen ändern. Dazu klickt man doppelt auf die zu bearbeitende Datei und geht dann wieder auf Tools-> Internationalization -> Internationalize dann erhält man z.B das folgende Fenster



Unter Key kann man nun den Schlüssel aussuchen, dem man einen neuen Wert zuweisen möchte und ein mögliches Kommentar hinzufügen welche bei einer späteren Bearbeitung oder bei einer Bearbeitung durch andere Personen hilfreich sein kann.

16.3. Noch zwei zusammenfassende Sätze:

Internationalisierung: Die Internationalisierung eines Programms bedeutet, dass es so entwickelt und vorbereitet wurde, dass es unter beliebig vielen Sprachen arbeitet.

Lokalisierung: Die Lokalisierung ist der Prozess des Anpassens auf eine bestimmte neue Sprache.

16.4. Quellen:

http://www.youtube.com/watch?v=JaftmZ6J7AE

http://www.youtube.com/watch?v=PSIR7ssjx0A&feature=related

http://netbeans.org/kb/docs/java/gui-automatic-i18n.html#newlocale

http://www.iks.hs-merseburg.de/~uschroet/Literatur/Java_Lit/JAVA_Insel/javainsel_13_004.htm

17. Web Application - David Hamm, 03.01.2011

17.1. Applets

17.1.1. Grundlagen

Ein Java-Applet ist ein Computerprogramm, das in der Programmiersprache Java verfasst wurde und normalerweise in einem Webbrowser (Abbildung 1), aber auch im Applet-Viewer (Abbildung 2) ausgeführt werden kann.



Abbildung 1



Abbildung 2

Sie wurden eingeführt, um Programme in Webseiten ablaufen lassen zu können, die im Webbrowser (auf der Client-Seite) arbeiten und direkt mit dem Benutzer interagieren können, ohne Daten über die Leitung zum Server versenden zu müssen (Abbildung 3).

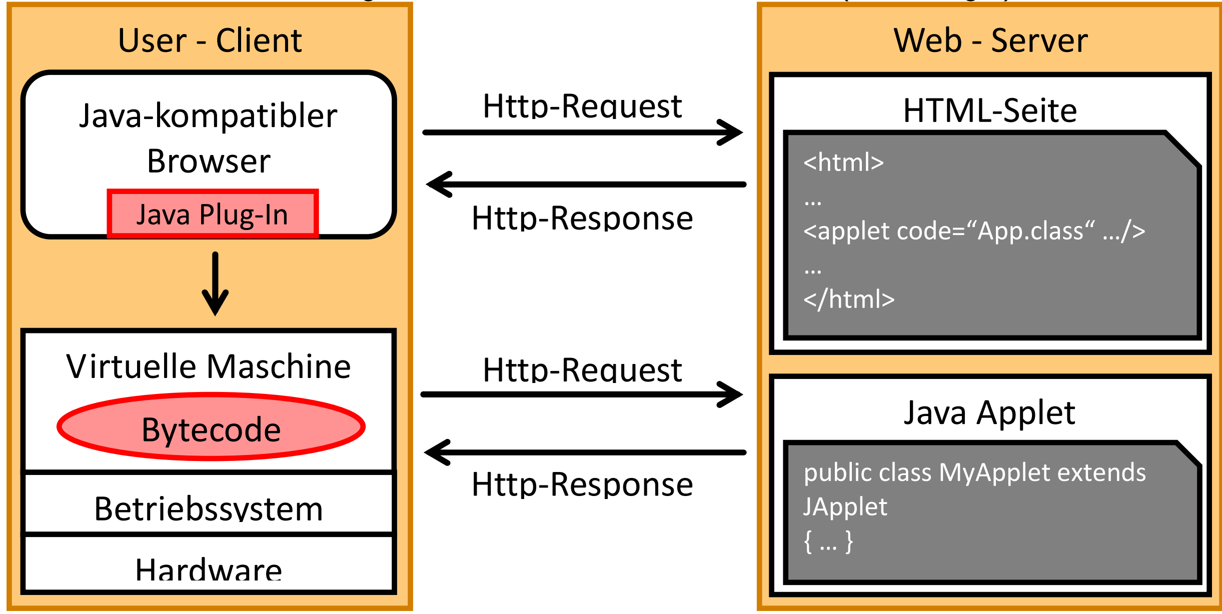


Abbildung 3

Java-Applets waren Ende der 1990er Jahre mit ein Grund für den Erfolg und die schnelle Verbreitung von Java.

17.1.2. Programmierung

Die Einbindung des Applets in den HTML-Code von Webseiten erfolgt mit dem **OBJECT**-Tag, dem **APPLET**-Tag oder dem **EMBED**-Tag. Alle Java-Applets werden von der Java-Klasse java.applet.Applet abgeleitet. Sie verfügen unter anderen über die folgenden Methoden, jedoch nicht über eine Methode main(), wie sie für Java-Anwendungen benötigt wird.

- init() - wird genau einmal aufgerufen, wenn das Applet erstmals geladen wird.
- start() - wird jedes Mal aufgerufen, wenn das Applet sichtbar wird.

- paint(…) - Zeichenmethode für die Anzeigefunktionen des Applet
- stop() - wird jedes Mal aufgerufen, wenn das Applet verdeckt wird, z. B. weil das Browser-Fenster von einem anderen Fenster überdeckt wird.
- destroy() - wird aufgerufen, wenn das Applet aus dem Hauptspeicher entladen wird.

Für die Programmierung der Benutzeroberfläche können die beiden Techniken Swing oder AWT verwendet werden. Jedoch ergibt sich die Einschränkung, dass das in die HTML-Oberfläche eingebettete Fenster nicht programmgesteuert geschlossen werden kann, man kann aber zusätzliche Fenster öffnen.

17.1.3. Sicherheit

Applets werden auf dem Rechner des Anwenders ausgeführt und stellen daher – wie jedes lokal ausführbare Programm – ein Sicherheitsrisiko dar (Java Drive-By Exploit).
Da die Applets in einer abgeschotteten Laufzeitumgebung (Sandbox) laufen, ist dieses Risiko jedoch gut beherrschbar. Ein Sicherheitsrisiko durch „böswillige“ Applets besteht nur, wenn die Sandbox fehlerhaft ist.
Applets die in dieser Sandbox laufen, können auf das lokale Dateisystem nicht zugreifen, können keine externen Programme starten und keine Netz-Verbindungen aufbauen.
Um diese Einschränkung zu umgehen kann ein Applet Signiert werden. Hierbei wird von unabhängigen Institutionen garantiert, dass das Applet keinen Schaden beim Nutzer verursacht.
Der Nutzer muss vor der Ausführung des Applets zustimmen, dass er dem Ersteller des Applets vertraut (Abbildung 4).

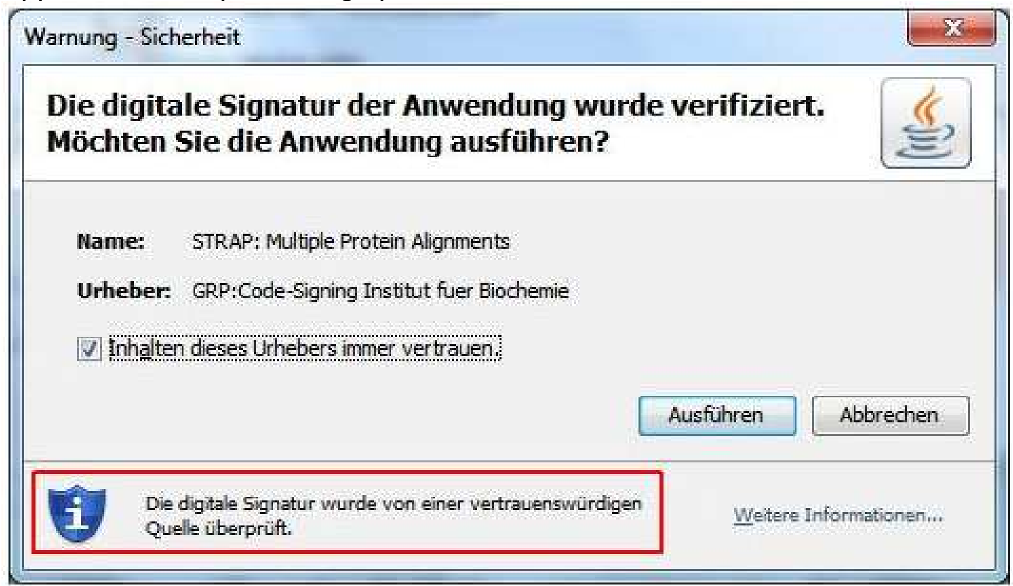


Abbildung 4

17.2. Java Server Pages

17.2.1. Grundlagen

JavaServer Pages, abgekürzt JSP, ist eine von Sun Microsystems entwickelte, auf JHTML basierende Web-Programmiersprache zur einfachen dynamischen Erzeugung von HTML- und XML-Ausgaben eines Webservers.
Sie erlaubt, Java-Code und spezielle JSP-Aktionen in HTML- oder XML-Seiten einzubetten. Dabei ermöglicht es die JSP-Syntax mittels spezieller XML-Tags (JSP-Aktionen) vordefinierte Funktionalität einzubinden. Diese JSP-Aktionen werden in so genannten Tag-Bibliotheken (Tag-Library) als Erweiterung der HTML- bzw. XML-Tags definiert.
JavaServer Pages werden unter Verwendung eines speziellen JSP-Compilers in Java-Quellcode umgewandelt. Dieser Quellcode, der einem Java-Servlet entspricht, wird im Anschluss durch den Java-Compiler in Bytecode umgewandelt. Die so erzeugten Java-

Klassen können dann von einem mit einer Java-Ausführungseinheit ausgestatteten Webserver ausgeführt werden (Abbildung 5).

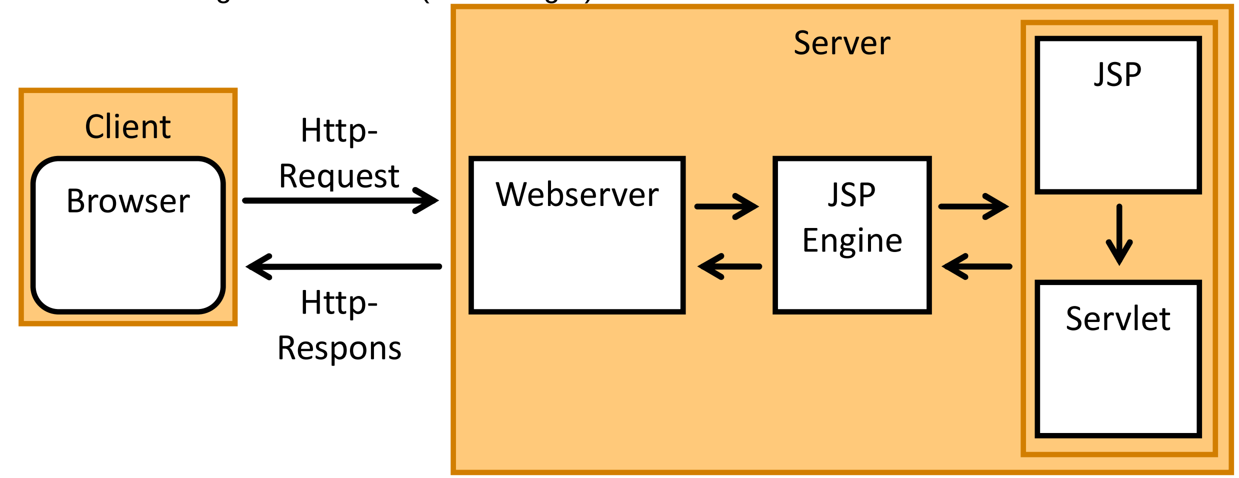


Abbildung 5

17.2.2. Elementtypen

Statischer Inhalt

Der **statische Inhalt** sind all jene Elemente, die vom Webserver in die HTTP-Response (Antwort) ohne Veränderung übernommen werden (z. B. HTML-Tags). So wäre ein normales HTML-Dokument gänzlich ohne JSP-Elemente, wie eingebetteten Java-Code oder JSP-Aktionen, eine gültige JSP; denn JSP erlaubt es zwar, dynamische Daten in statischen Inhalt einzufügen, erfordert dies aber nicht. Daraus folgt, dass (wie bei jeder Templatesprache) der Ersteller des statischen Inhalts (also des Designs) keinerlei Java-Kenntnisse benötigt; sobald HTML-Strukturen durch Java-Code erzeugt werden, geht die Verbindung jedoch verloren, und das Layout kann an diesen Stellen nur noch vom Programmierer geändert werden.
Da es sich bei einer JavaServer Page um eine dynamische Komponente handelt, kann diese aber auch gänzlich ohne statische Elemente auskommen. Auf diese Weise ist es möglich, über ein JSP-Dokument auch Binärdaten wie Bilder oder Töne zu erzeugen und an den Client zu senden.

Direktiven

Eine Direktive dient zum Übermitteln spezieller Seiteninformationen an den JSP-Compiler; dadurch kann man angeben, ob die JSP eine Tag-Library (kurz *Taglib*) einbindet oder wie im Fehlerfall weiter zu verfahren ist.
Die allgemeine Syntax für eine Direktive ist **<%@ ... %>**. Folgende Direktiven (mit den am weitesten verbreiteten Attributen) sind vorhanden:

Include

weist den JSP-Compiler an, den vollständigen Inhalt einer externen Datei in die Originaldatei zu kopieren. Diese Einbettung geschieht zur *Translation Time*, also statisch einmalig mit der Übersetzung der JSP in ein Servlet.

```
<%@ include file="BeispielDatei.ext" %>
```

Page

import

generiert ein Java-Import-Statement in der Datei

```
<%@ page import="java.util.*" %>
```

contentType

gibt die Art des Datei-Inhaltes an. Sollte dann eingesetzt werden, wenn man kein HTML benutzt oder nicht den voreingestellten Zeichensatz verwendet.

```
<%@ page contentType="text/html" %>
```

errorPage

gibt die Seite an, die im Fehlerfall angezeigt werden soll

```
<%@ page errorPage="ErrorHandler.jsp" %>
```

isErrorPage

gibt an ob diese Seite eine Error-Page ist oder nicht; wenn ja, ist das exception-Objekt verfügbar

```
<%@ page isErrorPage=false %><%-- die Seite ist keine Error-Page --%>
```

isThreadSafe

gibt an, ob das aus der JSP generierte Servlet threadsicher ist oder nicht

```
<%@ page isThreadSafe=true %><%-- eine threadsichere JSP --%>
```

Taglib

gibt an, dass eine Taglib verwendet werden soll. Es müssen ein Präfix und eine URI für die Taglib vergeben werden.

```
<%@ taglib prefix="MeinPrefix" uri="taglib/MeineTagLib.tld" %>
```

Standardvariablen

Die folgenden Variablen können in jeder JSP verwendet werden; Sie werden auch als implizite Objekte bezeichnet:

application

Anwendungsobjekt; lebt, solange der Server läuft.

config

eine Konfigurationsschnittstelle

out

JSPWriter, der die Daten in den HTTP-Response-Stream schreibt

page

das Servlet selbst

pageContext

eine Instanz des PageContext, welche die Daten der gesamten Seite enthält

request

das HTTP-Request-Objekt, also die Anfrage vom Client an den Server

response

das HTTP-Response-Objekt, also die Antwort vom Server an den Client

session

das HTTP-Session-Objekt. Es kann dazu benutzt werden, Information über den Benutzer von einem Request zum nächsten weiterzureichen.

Skriptelemente

Es gibt drei grundlegende Skriptelemente, die erlauben, Java-Code direkt in die JSP einzufügen:

- Ein Tag, das erlaubt, Code in die Klasse einzufügen. Dieses Tag kann dazu verwendet werden, Daten der Klasse festzulegen. Der Ausdruck wird in Member-Variablen der generierten Servlet-Klasse überführt.

```
<%! int serverInstanceVariable = 1; %>
```

- Ein Tag, das erlaubt, Code in die Methode `_jspService` des generierten Servlets einzufügen. Es werden also lokale Variablen daraus – Methoden können nicht erzeugt werden, hierzu muss die vorstehende Notation verwendet werden.

```
<% int localStackBasedVariable = 1; %>
```

- Ein Tag, das erlaubt, Code zu expandieren und direkt in die HTTP-Antwort zu schreiben. Das Semikolon wird hier nicht benutzt, da der Code als Ausdruck ausgewertet wird.

```
<%= "expanded inline data " + 1 %>
```

Deklarationen

Deklarationen dienen zur Definition von Variablen und Methoden, die von anderen Elementen in der JSP verwendet werden können. Deklarationen erzeugen keine Ausgabe innerhalb der JSP.

```
<%! int variableMeinerKlasse = 0; %>
```

Ausdrücke

Ausdrücke *(expressions)* werden dazu verwendet, Variablen oder Methoden direkt in den HTML- oder XML-Ausgabestrom zu integrieren.

```
Die Klassenvariable ist <%= variableMeinerKlasse %>
```

Skriptlets

JSP-Skriptlets können zur Implementierung der Ablauflogik sowie der Erzeugung der HTML- oder XML-Ausgabe eingesetzt werden. Der Skriptlet-Code wird innerhalb der Methode `_jspService` des generierten Servlets eingefügt.

```
<% int variable = 0; out.println("Der Wert der Variable ist: " +
variable); %>
```

Kommentare

Kommentare sind nur innerhalb der originalen JSP sichtbar, sie werden nicht in den Ausgabestrom geschrieben.

```
<%-- Kommentar innerhalb einer JSP --%>
```

17.2.3. Beispiel

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">

<%
String antwort = "";
String antwort_err = "";
Double celsius = 0.0;
Double kelvin = 0.0;

if (request.getParameter("submit") != null)
{
    try{
        celsius = Double.valueOf(request.getParameter("tcelsius"));
        if(celsius < -273.15)
            antwort_err = "Bei -273.15° ist schluss^^\nKälter gehts wirklich nicht.";
        else{
            kelvin = celsius + 273.15;
            antwort = String.valueOf(kelvin);
        }
    } catch (NumberFormatException e){
        if(request.getParameter("tcelsius").equals(""))
                antwort_err = "Ohne Eingabe gehts leider nicht.";
        else
            antwort_err = "Na Na :) Natürlich funktionieren hier nur Zahlen als Eingabe.";
    }
}
%>


<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Der Umrechner</title>
    </head>
    <body bgcolor="#74CCCC">
        <center>
        <h1>Der Umrechner</h1>
        <p>Hier haben Sie die Möglichkeit Temperaturen von Celsius in Kelvin umzurechenen:</p>

        <form action="" name="umrechner" method="GET">
          Celsius = <input type="text" name="tcelsius" value="<%=celsius%>">
          <br>
          <br>
          <input type="submit" name="submit" value="Berechnen">
        </form>
          <br>
          Kelvin = <input readonly="readonly" value="<%=antwort%>">
          <br>
          <%=antwort_err%>

        </center>
    </body>
</html>
```

In diesem Beispiel sehen wir einen einfachen Temperaturmaßeinheitenumrechner. Dabei wurde ein Skriptlet an den Anfang der JSP-Datei gestellt. Dieses Skriptlet berechnet uns beim klicken des Berechnen-Buttons, die Temperatur in Kelvin. Es wurde ganz normaler Java-Code eingefügt.
Im unteren Teil wird das HTML-Dokument aufgebaut, das dann mit den Werten gefüllt wird, welche das Skriptlet berechnet.

18. Projekt- und dependency management mit Maven - Holger Jansen, 03.01.2011

18.1. Was ist Maven

Maven ist ein von der Apache Foundation entwickeltes Tool für automatisiertes build und Dependency Management.
Seine besonderen Stärken liegen in automatisierten Tests, dem project build und dem bereits angesprochenem Dependency Management.
Des weiteren zeichnet es sich durch seine Plugin Architektur aus. Dies macht es sehr flexibel und somit für viele Anwendungsgebiete nutzbar. Genaueres dazu kann unter folgendem Link nachgelesen werden:

http://maven.apache.org/plugins/index.html
Dort findet sich auch eine kleine Auswahl an Plugins zum nachschauen und „spielen".

18.2. Warum Maven

In der Praxis wäre es zu umständlich alle Klassen von Hand anzugeben und mittels *javac* selber zu kompilieren. Diese Arbeit nimmt einem Maven ab und macht es möglich komplette Projekte automatisiert zu kompilieren, zu testen und anschließend in z.B. ein *.jar oder *.war Archiv zu packen und zu deployen.
Außerdem ist es komplett von der Konsole aus bedienbar und somit für das bauen von Anwendungen auf Servern ohne grafischer Oberfläche bestens geeignet.
Dies ist zwar auch schon mittels make oder ANT möglich, allerdings ist Maven wesentlich komfortabler in der Bedienung.
Außerdem ist Maven auch sehr gut in den gängigen IDEs wie Netbeans und eclipse integriert.
Weitere Features die für Maven sprechen können hier nachgelesen werden:
http://maven.apache.org/what-is-maven.html
http://maven.apache.org/maven-features.html

18.3. Die pom.xml

Pom steht für Project Object Model .
Die pom.xml ist die zentrale Konfigurationsdatei in Maven. Genau genommen ist die pom.xml die xml-Repräsentation des Projektes und dessen Einstellungen.
Maven nutzt diese um den kompletten Buildprozess zu steuern und alle nötigen

Informationen abzulegen. In ihr finden sich z.B:

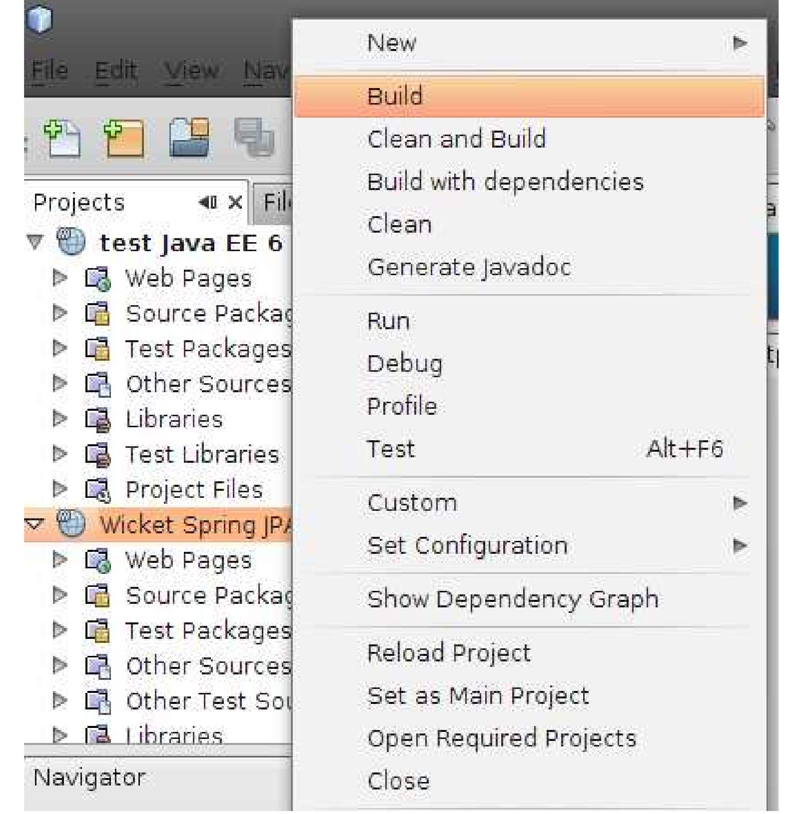
- Dependencies
- Repositories
- build Parameter
- Plugins
- Filtereinstellungen
- etc.

Alles was zur Beschreibung des Projektes, aus Sicht von Maven, nötig ist findet sich also dort. Aus Sicht von Maven muss allerdings ein Projekt keine Javaklassen oder ähnliches enthalten sondern lediglich eine valide pom.xml.
Die komplette pom.xml Doku ist auf der Maven Projektseite unter http://maven.apache.org/pom.html zu finden.

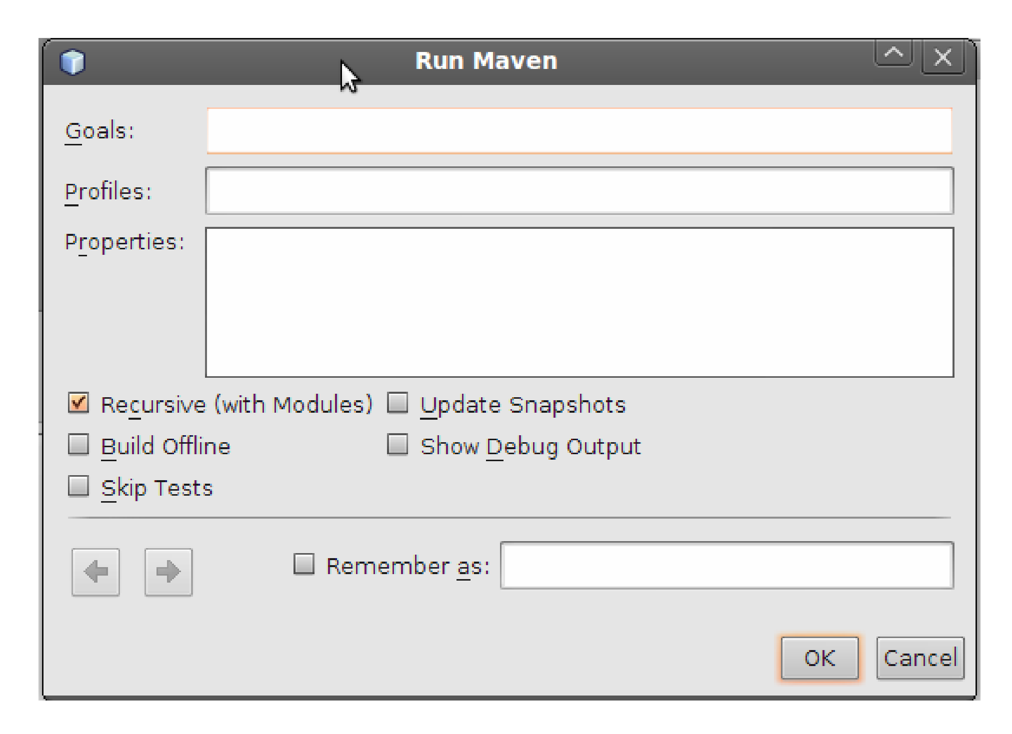
18.4. Welche Vorteile bringt mir Maven

18.4.1. Automatisierte Builds

Wie unter Punkt 2 bereits angesprochen wäre es in der Praxis zu unhandlich alle zu kompilierenden Dateien von Hand anzugeben. Deshalb unterstützt Maven das automatisierte bauen von Projekten. Um diesen Prozess auszulösen muss grundsätzlich erst einmal das Maven Kommando *„mvn“* aufgerufen werden. An dieses werden dann entsprechende *„goals“* angehangen. Die gängigsten Goals sind z.B. *clean, test, package,install …*
Es können beliebig viele Goals hintereinander ausgeführt werden. So macht es häufig Sinn vorher das Projekt zu *„cleanen“* um alle alten Klassen etc. zu löschen.
Um also eine Anwendung zu löschen, anschließend neu zu bauen und in ein Archiv, wie in der pom konfiguriert zu packen wäre das Kommando: *„mvn clean package“*
Eine Liste aller verfügbaren Goals und Infos zum Lifecycle Management finden Sie hier:
http://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html
Der gleiche Effekt kann in Netbeans durch einen Rechtsklick auf den Projektnamen erreicht werden.



In diesem Kontextmenü sind die gängigsten Goals direkt enthalten und können mittels Klick ausgeführt werden. Netbeans setzt dies dann in die entsprechenden Maven Kommandos um. Unter dem Punkt Custom können noch weitere Goals mit Parametern konfiguriert werden.



18.4.2. Convention over Configuration (CoC)

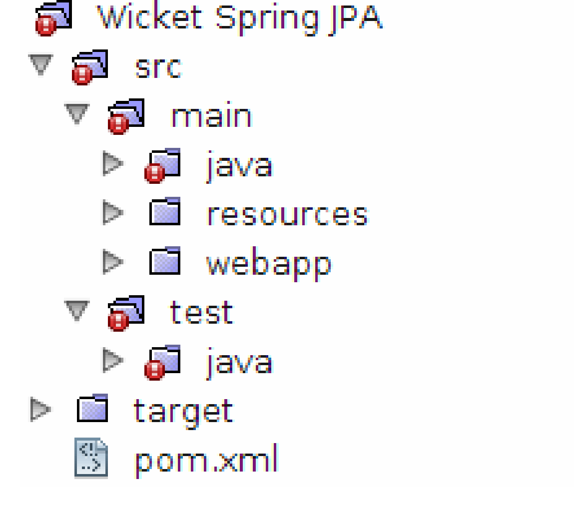
CoC bezeichnet ein Software-Paradigma.
Dieses Paradigma kommt auch bei Maven zum Einsatz und ist einer der Gründe warum es den Umgang und die Einarbeitung in Maven so einfach macht.
Allgemeines zu Convention over Configuration können hier entnommen werden:
http://de.wikipedia.org/wiki/Konvention_vor_Konfiguration
http://en.wikipedia.org/wiki/Convention_over_configuration

Explizit auf Maven bezogen wird dies mit unter anderem folgenden Zielen eingesetzt:

- Komplexität von Konfigurationen zu vereinfachen
- Schnellerer und einfacher Durchblick in Projekten
  ==> höhere Wartbarkeit
- Dennoch dem Entwickler alle Freiheiten lassen das Projekt anders zu konfigurieren

Mit dem Hauptziel der ==> Kosteneinsparung

In der Praxis resultiert daraus eine vorgegebene Paket-Hierarchie/Projektstruktur



Dies wäre eine Beispielstruktur für eine Webanwendung unter Maven

src: Hier befinden sich allgemein erst mal alle Eingabedateien
|- **main:** Alle Projektspezifischen Dateien
| |- **java:** Alle Javaquellcodes

| |- **resources:** Resourcen wie z.B. Properties
| |- **webapp:** Alle Webapp resourcen wie *.html oder *.css Dateien
test: Hier wird die gleiche Struktur wie unter src aufgebaut. Allerdings befinden sich dort die Unit-tests zu den Klassen im src Verzeichnis.
target: Hier befindet sich anschließend das fertig kompilierte Programm.

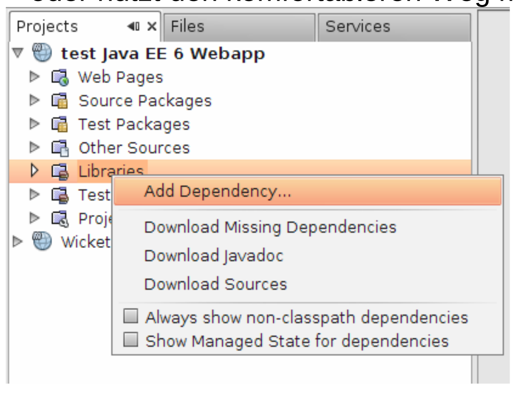
Dadurch kann ein Entwickler sich schneller in die Struktur eines Projektes einarbeiten da er auf Anhieb weiß wo er welche Dateien finden kann.

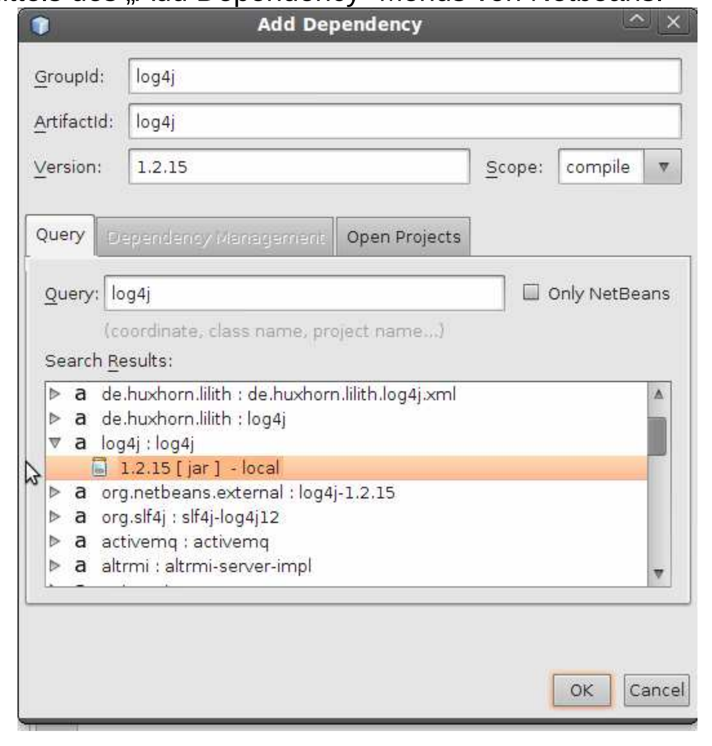
18.4.3. Dependency Management

Dependencies bezeichnen Abhängigkeiten. Sprich Libraries. Diese sind in größeren Javaprojekten nicht weg zu denken. Man braucht z.B. um eine Verbindung zu einer Datenbank aufzubauen einen jdbc-driver oder um Logging mittels log4J zu nutzen eine Library. Diese werden in einem Archiv im *.jar Format zur Verfügung gestellt. Ohne Maven müsste man als Entwickler sich selber das jar besorgen, dieses ins Projekt einbinden und richtig platzieren. Diese Arbeit nimmt einen Maven ab. Mittels Maven fügt man nun entweder händisch in die pom.xml die entsprechende dependency ein. z.B.

```
<dependency>
  <groupId>log4j</groupId>
  <artifactId>log4j</artifactId>
</dependency>
```

oder nutzt den komfortableren Weg mittels des „Add Dependency“ Menüs von Netbeans.





Netbeans fügt dann den obigen Code in die pom.xml ein.
Dies hat auch den Vorteil das die Bibliotheken nicht mit ins Versionierungssytem (CVS/Subversion) eingecheckt werden müssen, da alle Bibliotheken zentral in Repositories zu Verfügung stehen. So können auch Versionskonflikte unter den Bibliotheken vermieden werden.
Während des compile-Vorgang schaut Maven im Hintergrund auf den in der pom konfigurierten Respositories nach ob es die gewünschte Dependency finden kann. Ist kein

Repository hinterlegt greift Maven auf das default Maven Respository zurück und lädt diese herunter und fügt sie dem Projekt hinzu.
Für die Meisten Anwendungen reicht eigntlich das default Maven Repository.
Sollte man die gewünschte Library nicht finden kann man diese natürlich auch in seinem lokalen Repositroy hinterlegen. Zu finden ist dieses unter:

- **Linux:** ~/.m2
- **Windows:** C:\Documents and Settings\USER\.m2

Weiterführende Infos dazu gibt es hier:
http://maven.apache.org/guides/introduction/introduction-to-repositories.html
http://docs.codehaus.org/display/GEOT/2.5.2+Maven+Local+Settings+and+Repository

Alternativ und in Firmen gängigere Praxis ist das aufsetzen eines eigenen Repository Servers wie z.B. Nexus von Sonartype http://nexus.sonatype.org/ wo Firmeneigene Projekte bereit gestellt werden. Alles weitere zum Dependency Management kann der Maven Doku unter http://maven.apache.org/guides/introduction/introduction-to-dependency-mechanism.html entnommen werden.

18.4.4. Automatisierte Tests

Ein weiteres Feature von Maven ist das automatisierte ausführen von Unit-Tests im Anschluss an den Compile Vorgang. Dazu schaut Maven im Verzeichnis „test“ nach und führt alle Unit-Test die es dort findet aus. Man baut dazu idealerweise die selbe Verzeichnisstruktur wie im Verzeichnis „src“ auf. Somit kann Maven auch automatisch alles von alleine finden und zuordnen.
Hierzu muss nicht gesondert das Goal *test* aufgerufen werden. Dies ist nur zum ausführen der Tests ohne Packen der Anwendung.
Realisiert und konfiguriert wird dies durch das „Surfire-plugin“ welches standardmäßig in Maven enthalten ist. Dieses generiert auch Protokolle im Anschluss an die Tests wo die Ergebnisse dieser nachgelesen werden können. Zu finden sind diese anschließend im Verzeichnis *„${base.dir}/target/surfire-reports/“*.
Die Besonderheit an diesem Plugin ist, dass wenn ein Test fehl schlägt, das komplette Projekt nicht gebaut wird und der Buildvorgang nach Beendigung aller Testcases mit einem *„BUILD FAILURE“* abgebrochen wird.

```
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.852 sec
Running de.hsnr.seminar.dao.jpa.TestEventDaoJPA
Tests run: 5, Failures: 0, Errors: 5, Skipped: 0, Time elapsed: 0.059 sec <<< FAILURE!

Results :

Tests in error:
  testFindAll(de.hsnr.seminar.dao.jpa.TestEventDaoJPA)
  testCountAll(de.hsnr.seminar.dao.jpa.TestEventDaoJPA)
  testDelete(de.hsnr.seminar.dao.jpa.TestEventDaoJPA)
  testLoad(de.hsnr.seminar.dao.jpa.TestEventDaoJPA)
  testSave(de.hsnr.seminar.dao.jpa.TestEventDaoJPA)

Tests run: 7, Failures: 0, Errors: 5, Skipped: 0

[INFO] ------------------------------------------------------------------------
[ERROR] BUILD FAILURE
[INFO] ------------------------------------------------------------------------
[INFO] There are test failures.

Please refer to /home/hoschy/NetBeansProjects/example/target/surefire-reports for the individual test results.
[INFO] ------------------------------------------------------------------------
[INFO] For more information, run Maven with the -e switch
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 41 seconds
[INFO] Finished at: Sun Jan 02 23:45:24 CET 2011
[INFO] Final Memory: 24M/174M
[INFO] ------------------------------------------------------------------------
```

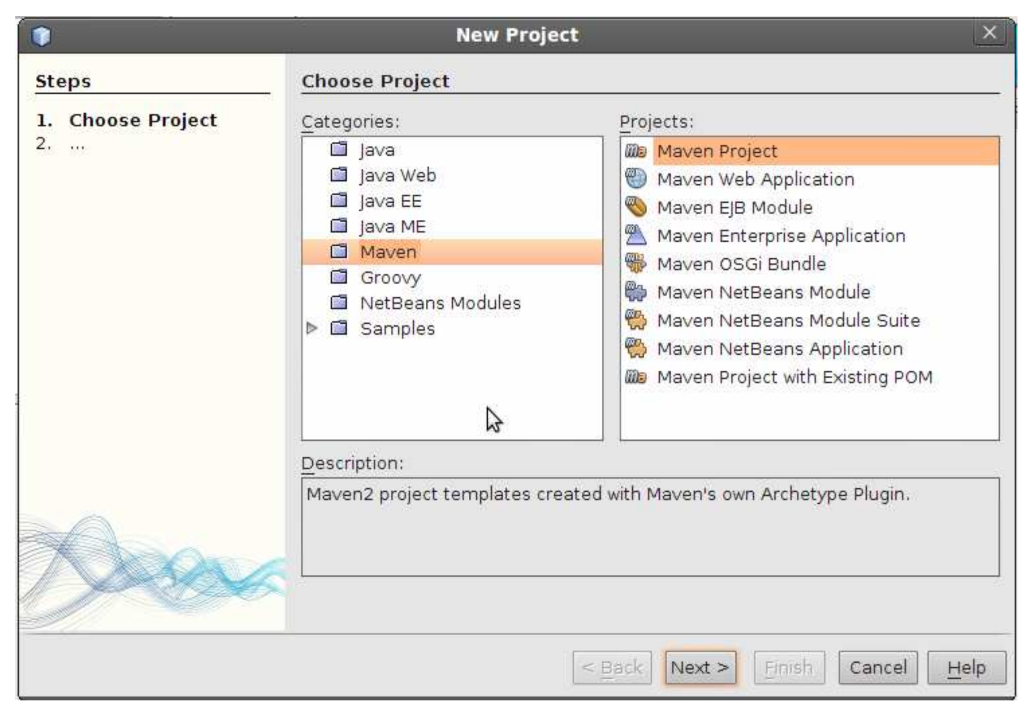
Der Sinn dahinter ist es nur Projekte die auch erfolgreich getestet werden können überhaupt zu bauen und zu verbreiten. Dies trägt erheblich zur Qualitätssicherung der Software bei.

Vorallem werden durch diesen Mechanismus auch Entwicklungsmethoden wie „Test-Driven-Development“ sehr gut unterstützt.
Natürlich kann man auch die Unit-Tests bewusst abschalten bzw. überspringen. Dies ist zwar eigentlich nicht der Weg wie Maven es vorsieht kann aber manchmal sehr nützlich sein. Z.B. wenn eine Anwendung/Test von einem Service abhängig ist der in diesem Moment nicht erreichbar ist oder man weiß das einer der Tests nicht korrekt funktioniert. Realisieren kann man dies mittels folgender Parameter die einfach an das Kommando angehängt werden.
-Dmaven.test.skip=true (compile und Test skip)
-DskipTests=true (nur Test skip)
Dies kann in Netbeans durch ein Custom Goal ebenfalls erreicht werden. Siehe dazu Punkt 4.1 Weitere Informationen sowie einige weiterführende Beispiele sind unter http://maven.apache.org/plugins/maven-surefire-plugin/ zu finden.

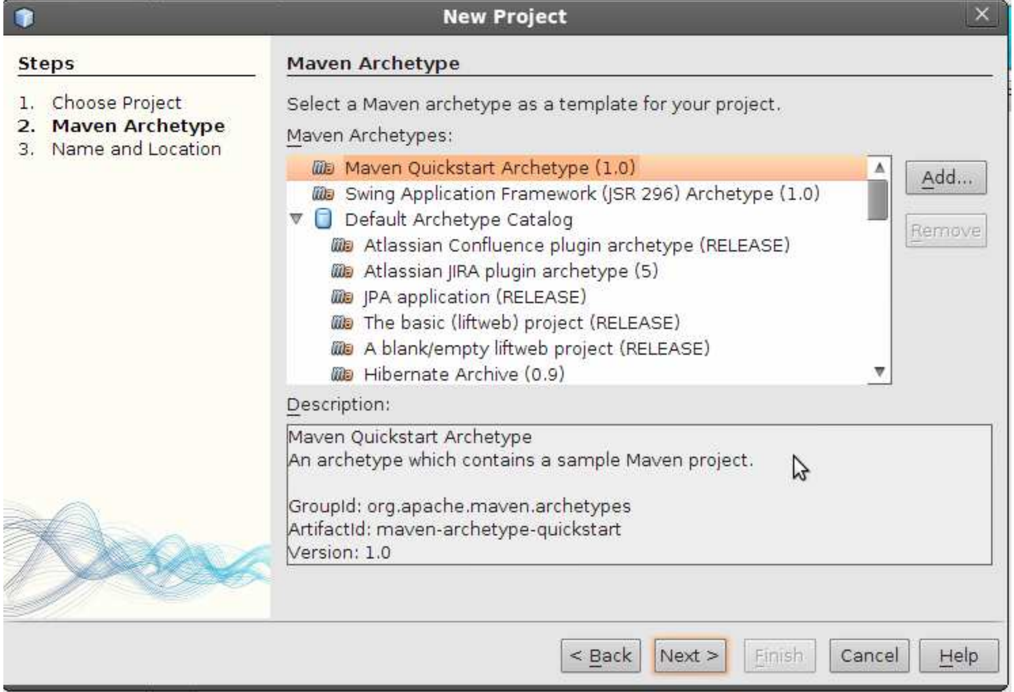
18.4.5. Archtypes

Archtype ist ein in Maven enthaltenes Projekt Templating Toolkit.
Umgangssprachlich werden diese Vorlagen deshalb auch Archtypes genannt.
Archtypes geben einem die Möglichkeit Projektstrukturen direkt nach z.B. Firmenvorgaben oder Best Practice zu generieren. Es wird also eine komplette Projektstruktur und je nach Archtype direkt auch einige Basisklassen etc. erstellt.
Eingeleitet wird dieser Vorgang durch das Kommando
„mvn archetype:generate“
Mit diesem Kommando wird man auf der Konsole anschließend durch einen Dialog geleitet welcher einem bei der Erstellung des Projektes Hilft. Dieser fragt z.B. die gewünschte Paket-Hierarchie sowie den Projektnamen und den Typ des Projekts ab.
Man kann natürlich auch direkt einen Archtype angeben.
Mit dem Kommando
„mvn archetype:generate -DarchetypeArtifactId=maven-archetype-webapp“
würde man also eine Webapp nach Maven Vorgaben generieren. So kann dieser Dialog abgekürzt werden. Man kann auch direkt alle nötigen Parameter übergeben und so diesen Dialog vollständig umgehen. Ein Beispiel welches dies Zeigt wäre z.B.
„mvn archetype:generate -B -DarchetypeCatalog=http://legup.googlecode.com/svn/repo/archetype-catalog.xml -DarchetypeArtifactId=wicket-spring-jpa-archetype -DarchetypeGroupId=com.jweekend -DarchetypeVersion=0.8.4 -DgroupId=de.hsnr.seminar -DartifactId=example -Dversion=1.0-SNAPSHOT -Dpackage=de.hsnr.seminar“
Dies ist ein vorgefertigter archtype zur Erstellung eines Projektes mit Spring, JPA, Hibernate und Wicket. Dies wäre ein typischer Archtype für eine moderne Webapp wie sie Täglich gebaut wird. Dieser generiert einem die komplette Ordnerstruktur inklusive einiger Startklassen, Unit-Tests und allen benötigten Dependencies etc. . Probieren Sie es doch einfach mal aus!
(ACHTUNG: Dauert beim ersten ausführen recht lange da zuerst alle benötigten Dependencies heruntergeladen werden müssen)

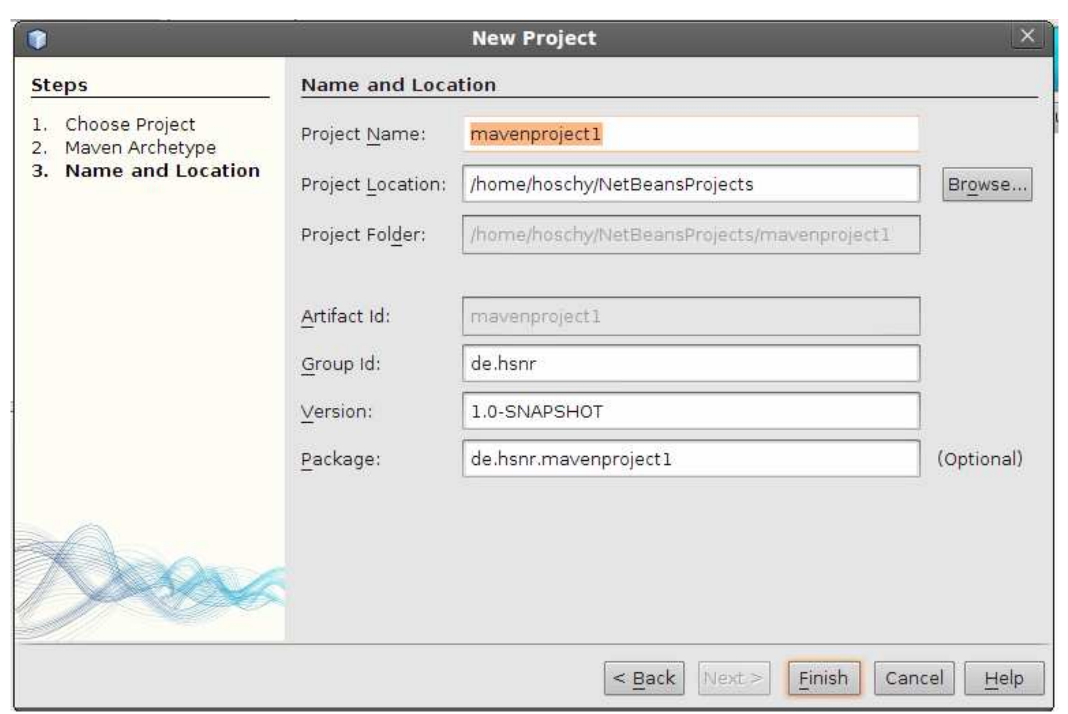
In Netbeans versteckt sich diese Funktionalität hinter dem New Project Wizzard.



Wählt man nun nur Maven Project gelang man nun zu folgendem Dialog:



Dies ist lediglich eine graphische Darstellung des Dialoges wie er auch auf der Konsole zu sehen ist. Wie man sieht kann man zwischen vielen verschiedenen Archtypes wählen. Maven Quickstart wäre der dabei der einfachste Archtype. Dieser generiert lediglich die Projekt Grundstruktur und eine Mainklasse.



Anschließend an diesen Dialog wird man nun wie gewohnt nach dem Projektnamen sowie einigen weiteren Maven spezifischen Parametern wie groupId oder der Version gefragt. Anschließend generiert Maven das Projekt und Sie können anfangen zu programmieren. Weitere Informationen sowie die Archtypedokumentation finden Sie hier:
http://maven.apache.org/guides/introduction/introduction-to-archetypes.html

18.5. Resümee

Abschließend kann man sagen das die Einarbeitung in Maven durchaus Sinn macht da es einem den Programmieralltag erheblich erleichtert. Es ermöglicht einem einfache Einarbeitung in Projekte sowie gute Mechanismen zur Qualitätssicherung. Der Konfigurationsaufwand wird minimiert und das Dependency Management nimmt einem eine Menge an Arbeit ab. Ebenso können in der Wirtschaft durch die obigen Vorteile erhebliche Kosten eingespart werden. Maven ist also momentan der modernste und komfortabelste Weg Javaprojekte zu bauen und zu verwalten.

Nicht Teil des Vortrags aber dennoch interessant & hilfreich:

18.6. Installationsanleitung

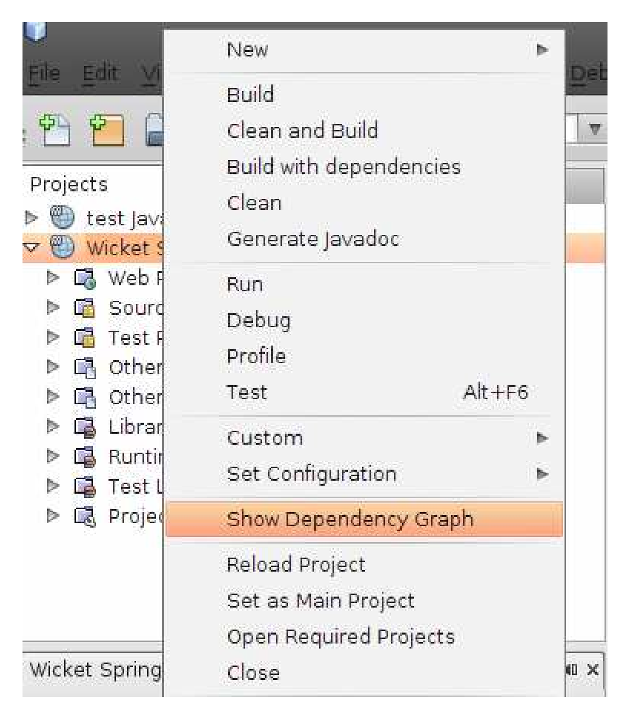
http://maven.apache.org/download.html
Am Ende der Seite.

18.7. Filtereinstellungen

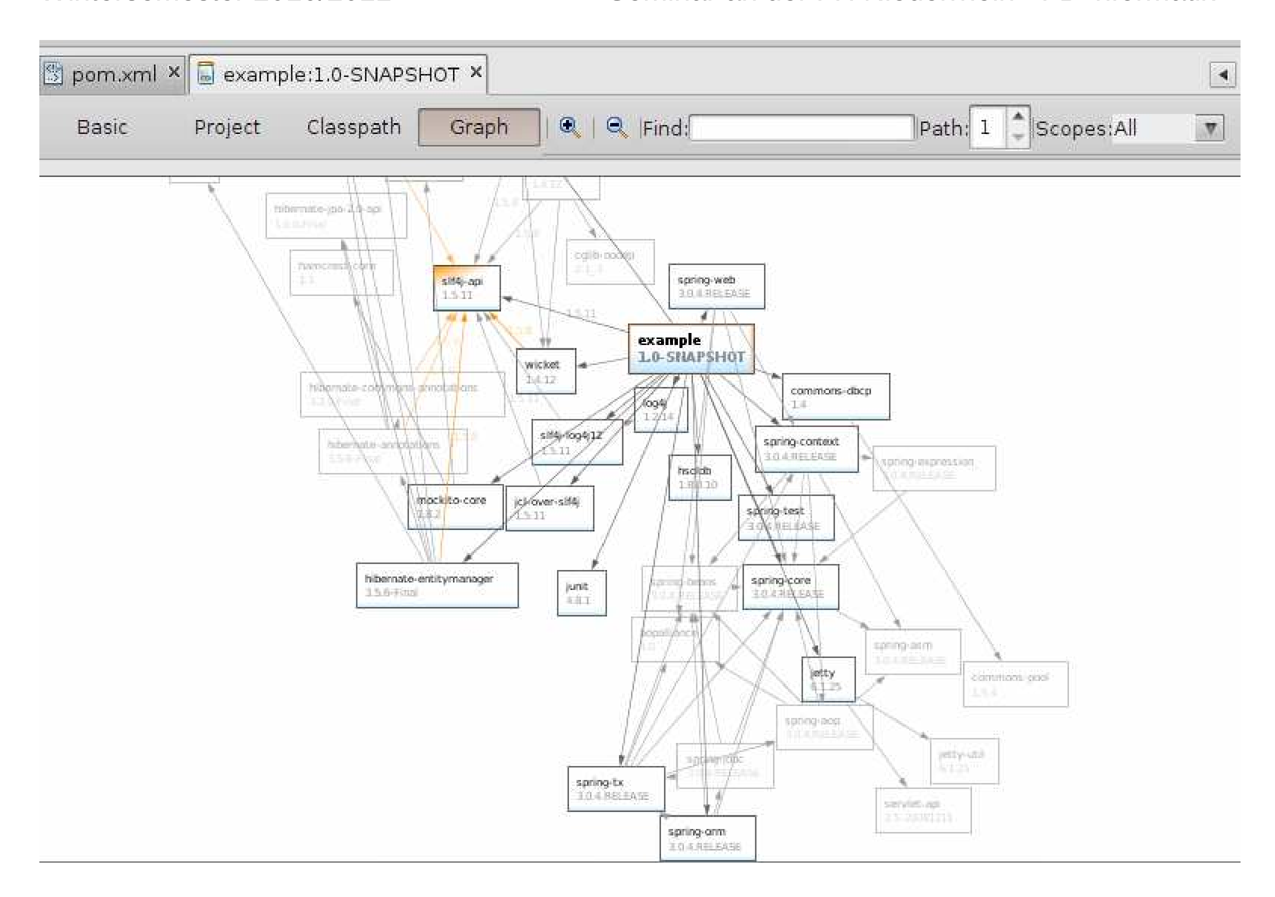
Sehr nützlich wenn Anwendungen auf unterschiedlichen Systemen deployed werden sollen. Durch Filtereinstellungen können verschiedene Properties für Systeme hinterlegt werden. Durch Angabe des entsprechenden Profils setzt Maven in alle Konfigurationsdateien die Werte aus den Properties des entsprechenden Systems ein. Somit können Anwendungen auf einfachen Weg für mehrere Syteme konfiguriert werden ohne dass alle Konfigurationsdateien eines Projektes geändert werden müssen sondern nur eine zentrale Datei pro System. Wie dies im Detail funktioniert entnehmen Sie bitte der Maven Doku:
http://maven.apache.org/guides/mini/guide-building-for-different-environments.html
http://maven.apache.org/guides/introduction/introduction-to-profiles.html
http://maven.apache.org/plugins/maven-resources-plugin/examples/filter.html

18.8. Dependency Graph

Der dependency Graph ist in Feature in Netbeans welches einem ermöglicht die dependencies eines Projekts graphisch darzustellen. Dies kann sehr nützlich sein da häufig Libraries auch Abhängigkeiten zu anderen Libraries haben und diese Abhängigkeiten ebenfalls dargestellt werden. So bekommt man einen guten Überblick welche Abhängigkeiten ein Projekt wirklich hat. Probieren Sie es aus. Sie werden überrascht sein. Zu erreichen ist dieser über einen Rechtsklick auf den Projektnamen → Show Dependency Graph



Unter den Reitern „Basic“ „Project“ „Classpath“ werden noch Weitere Eigenschaften des Projektes dargestellt. So ist dies besser auf einen Blick erfassbar als in der pom.xml.



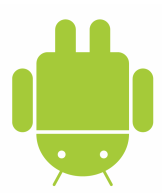
18.9. Andere nützliche Links

Getting started guide.
http://maven.apache.org/guides/getting-started/index.html
Maven in 5 Minutes (Kurzform des Getting started guide)
http://maven.apache.org/guides/getting-started/maven-in-five-minutes.html

19. Mobile Application mit Android - Dominic Scharmann, 10.01.2010

Vorwort
Dieses Kurzskript soll nicht dazu dienen, ihnen komplett die Grundlagen von Android zu vermitteln. Hier wird nur leicht an der Oberfläche gekratzt, um einen kurzen Einblick in die Android-Welt zu geben und um Ihr Interesse an diesem großen Open Source Projekt zu wecken.

19.1. Was ist Android?

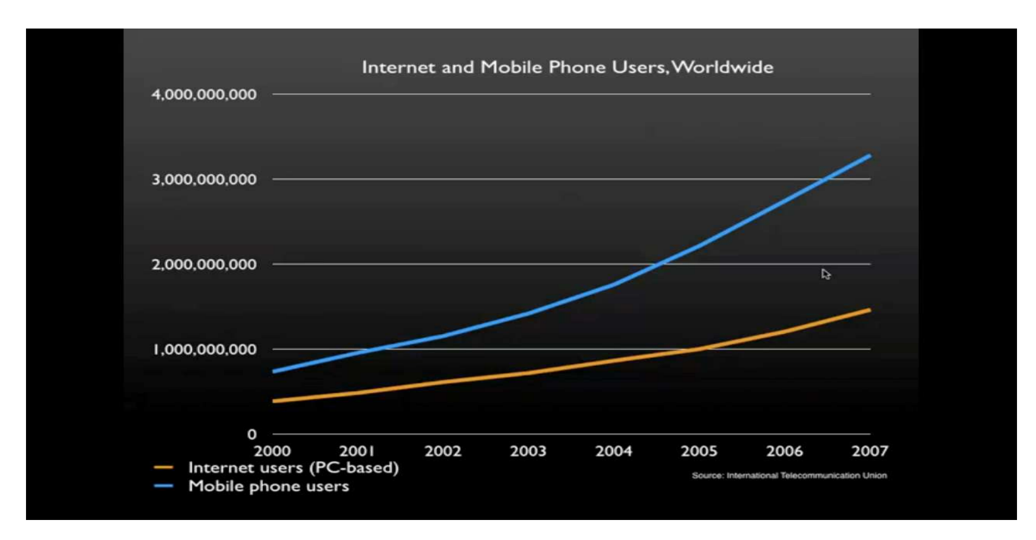


Android ist ein Betriebssystem für mobile Geräte – zum Beispiel für Smartphones, tabletPCs oder Netbooks. Android wurde 2003 gegründet, aber im Sommer 2005 von Google aufgekauft. Im November 2007 gab Google offiziell bekannt, gemeinsam mit 33 anderen Mitgliedern der Open Handset Alliance ein Betriebssystem für Mobiletelefone namens Android zu entwickeln. Ende 2008 war die erste Android Version verfügbar.

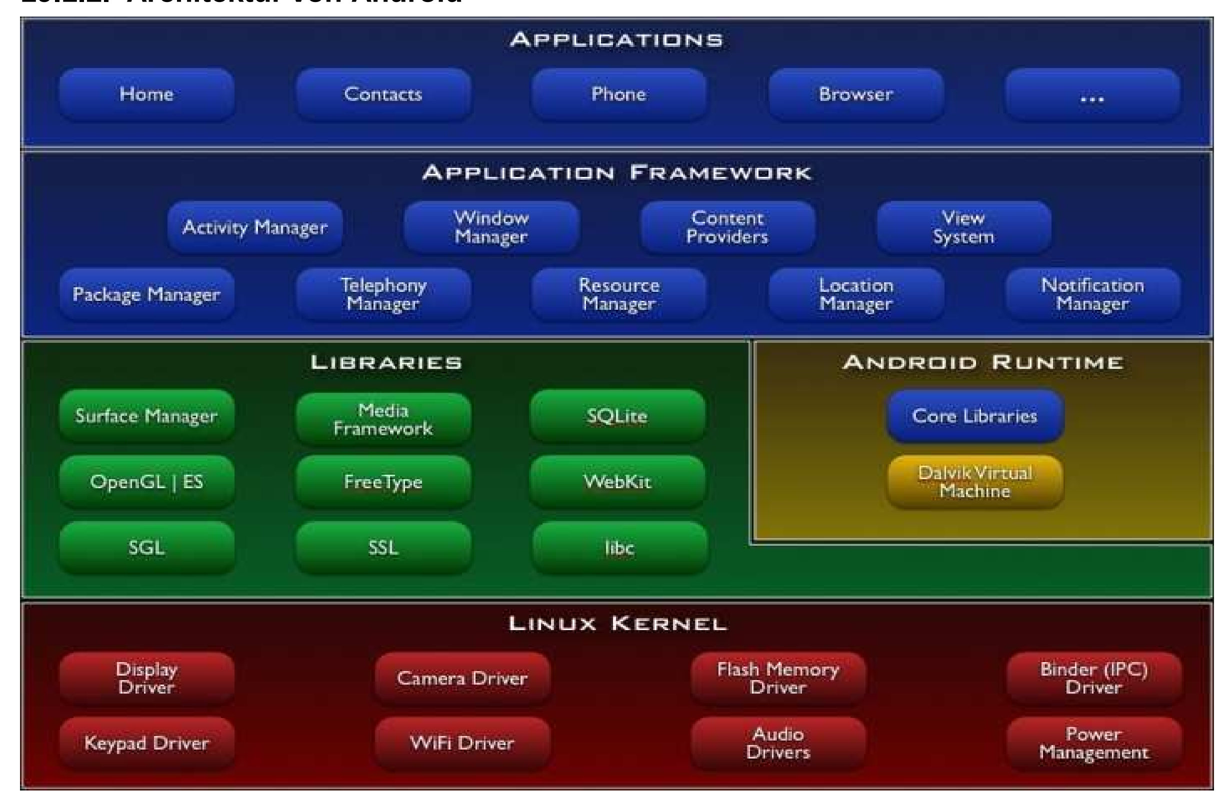
Android basiert auf dem Linux-Kernel 2.6 ist freie Software und quelloffen. Das bedeutet, dass Mobiletelefonhersteller keine Lizenzgebühren zahlen müssen, wenn sie Android als Betriebssystem auf ihren Geräten verwenden. Im Dezember gab Google an, dass pro Tag 300.000 Android Mobiletelefone verkauft werden. Zurzeit besitzt Android als Smartphone-Betriebssystem einen weltweiten Marktanteil von 25 %, zum Vergleich: Apple`s iPohne OS hat einen Marktanteil von 26 %.

19.1.1. Warum stellt Google Android kostenlos zur Verfügung?

Die Grafik zeigt das weltweite Verhältnis zwischen Handy-Nutzern und Internet fähigen PC-Nutzern. Ende 2007 gab es mehr als 3 Mrd. Handy-Nutzer, wenn man nun davon ausgeht, dass 10% in früher Zukunft mit ihrem Smartphone online gehen, ist das ein Markt von 300 Millionen Nutzer. Da Google mit Werbung/Anzeigen/Daten Geld verdient, ist dieser Markt für Google sehr lukrativ. Wenn man bedenkt, welche Flut an Daten Google zur Verfügung steht, könnte bei Ausnutzung dieser Daten eine genau an die Nutzer angepasste Werbung auf dem Smartphone erscheinen. Diese Leistung könnte sich Google von Werbetreibenden gut bezahlen lassen und somit einen recht klaren Wettbewerbsvorteil ausnutzen. So ist es auch möglich, in Apps Werbung anzeigen zu lassen.



19.1.2. Architektur von Android



Linux Kernel
Der aktuelle Linux-Kernel ist die Version 2.6. Dieser bildet eine gute Betriebssystemgrundlage, indem er die erforderlichen Hardwaretreiber zur Verfügung stellt – wie zum Beispiel verschiedene Gerätetreiber (Bildschirm, Kamera, WiFi etc.). Zudem übernimmt der Kernel die Energieverwaltung, Prozessverwaltung, usw.

Android Runtime (Laufzeitumgebung)
Wird eine Anwendung gestartet, läuft diese in ihrer eigenen virtuellen Maschine. In Android wird die Dalvik Virtual Machine (DVM) genutzt. Diese DVM ist so klein und performant, dass Android für jede weitere gestartete Anwendung eine neue DVM spendiert. Dies kann unter Umständen zwar extra Ressourcen verbrauchen, dennoch bringt es einen großen Vorteil in punkto Sicherheit und Verfügbarkeit, da sich die Anwendungen keinen gemeinsamen Speicher teilen, kann ein sterbender Prozess keine weiteren Anwendungen zum Absturz bringen.

Libraries (Standardbibliotheken)
Diese C/C++ Bibliotheken stellen alle erforderlichen Funktionalitäten bereit, die eine Android Anwendung erfordert.

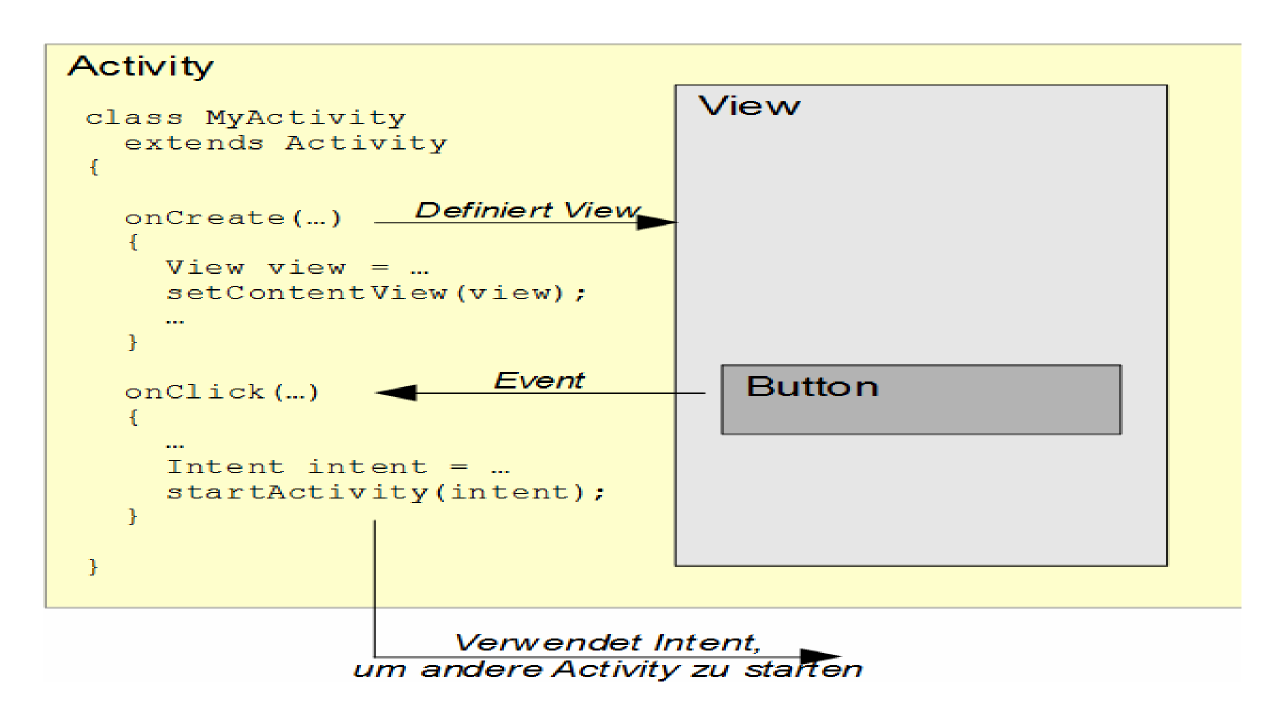
Application Framework (Programmierschnittstelle/Anwendungsrahmen)
Diese Schicht in der Android-Architektur ist für den Entwickler die interessanteste. Sie stellt verschiedene Programmierschnittstellen bereit, die zur Kommunikation zwischen Anwendungen oder zwischen Benutzer und Anwendungen erforderlich sind, wie z.B.:
- View System, zum Anzeigen von Listen, Textfelder, Buttons etc.
- Conten Provider, zum Austausch von Daten zwischen Anwendungen
- Resoruce Manager, verwaltet Ressourcen wie z.B. Grafiken, Sounds etc.
- Activity Manager**,** verwaltet den Lebenszyklus einer Anwendung.

Applications (Anwendungen)
In dieser Schicht liegen die eigentlichen Anwendungen (Apps), da Android „offen“ ist, können selbst die Standardanwendungen komplett ausgetauscht werden.

19.1.3. Komponenten in Android

Crash-Kurs
Was sind Software Komponenten?
Eine Komponente in der Software-Entwicklung ist ein Teil einer Software. Das Interface der Komponenten ist eine Schnittstelle zum Rest der Software. Dabei kann eine Komponente gemäß eines Composition Standards unabhängig zu anderen Komponenten ausgeführt werden. Das Komponentenmodell legt einen Rahmen für die Entwicklung und Ausführung von Komponenten fest.

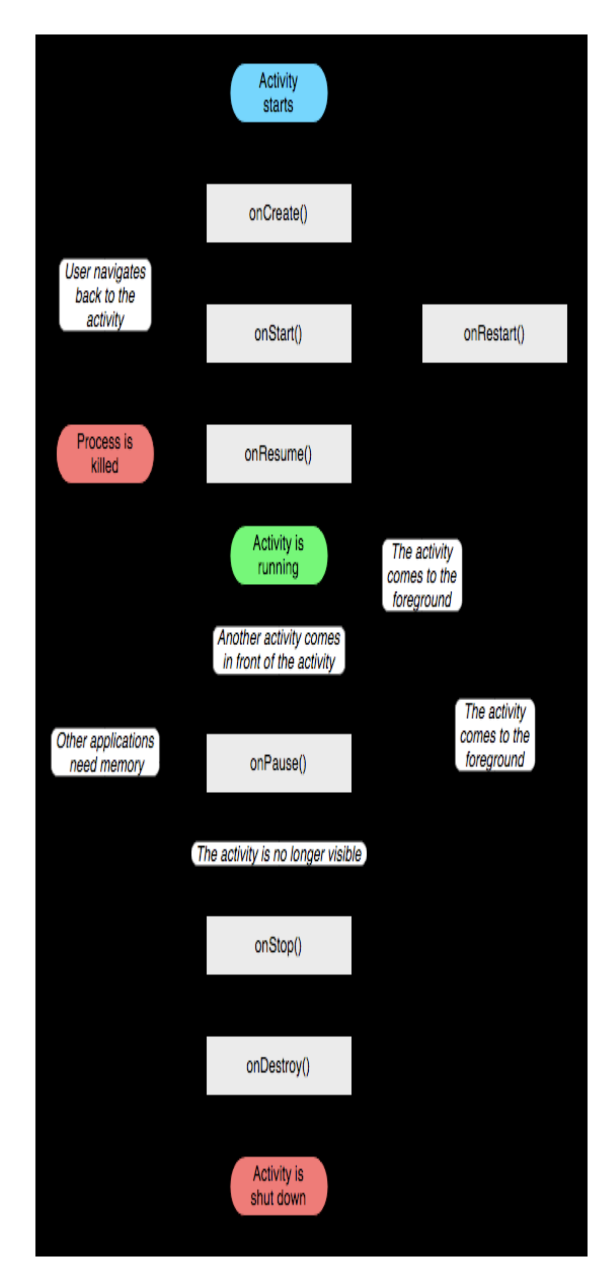
In Android wird ein einfaches Komponentenmodell definiert: Activity, Content Provider, Broadcast Receivers, Services. Die Verbindung der Komponenten untereinander wird über „Intents" realisiert. Dabei ist die wichtigste Komponente in Android die Activity, die hier kurz dargestellt ist:



Eine Activity wird gebraucht, um eine Oberfläche anzuzeigen. Activitys sind sichtbar und können miteinander zu einer komplexen Anwendung verknüpft werden. Sie kümmert sich, indem sie Daten auf dem Screen darstellt und Anwendereingaben entgegennimmt, um die Kommunikation mit der Anwendung und dem Endbenutzer.

19.1.4. Activity Lifecycle

Mobile Geräte haben limitierte Ressourcen. Im Falle knapper Ressourcen müssen Prozesse beendet werden. Daher greift das Android-Betriebssystem in die Lebensdauer von Prozessen ein und kann jederzeit inaktive Activitys (Anwendungen, die nicht im Vordergrund sichtbar sind) eigenständig beenden. Daher hat jede Activity einen Lifecycle („Lebenszyklus"), der vom Betriebssystem kontrolliert wird. Durch das eigenständige Beenden von Prozessen ist die Gefahr von Datenverlust sehr hoch. Natürlich gibt es verschiedene Implementierungsmuster, wie man das plötzliche Beenden eines Prozess umgeht und wie man Datenverlust vermeidet.



OnCreate(...)
Kann als Konstruktor angesehen werden, dort wird die Activity erzeugt mit allen Initialisierungen (Menüs, Layouts, etc.).

OnStart()
Wird aufgerufen, wenn die Activity neu erzeugt wird oder eine „alte“ Activity zurück in den Vordergrund gerufen wird.

OnResume()
Wird aufgerufen, wenn die Activity nur teilweise verdeckt war und nun wieder Vollständig angezeigt werden soll.

OnPause()
Die Activity ist hier nur teilweise sichtbar (sie wird z. B. von einer anderen Activity überdeckt) und reagiert nicht mehr auf Benutzereingaben.

OnStop()
Wird aufgerufen, wenn die Activity vollständig in den Hintergrund kommt und beendet wird. Sie wird aber auf einen „Activity-Stack“ gelegt, falls sie nochmal aufgerufen wird.

OnRestart()
Wird aufgerufen, wenn z. B. der Benutzer die „Zurück-Taste“ gedrückt hat und eine bereits vorhandene Activity aufgerufen wird.

OnDestroy()
Hier wird die Activity beendet und alle Ressourcen werden freigegeben. Kann als Destruktor angesehen werden.

19.2. Android in Netbeans einrichten

19.2.1. Android SDK

Bevor Sie mit der Entwicklung von Android-Anwendungen in Netbeans starten, müssen Sie sich die Android SDK runterladen.

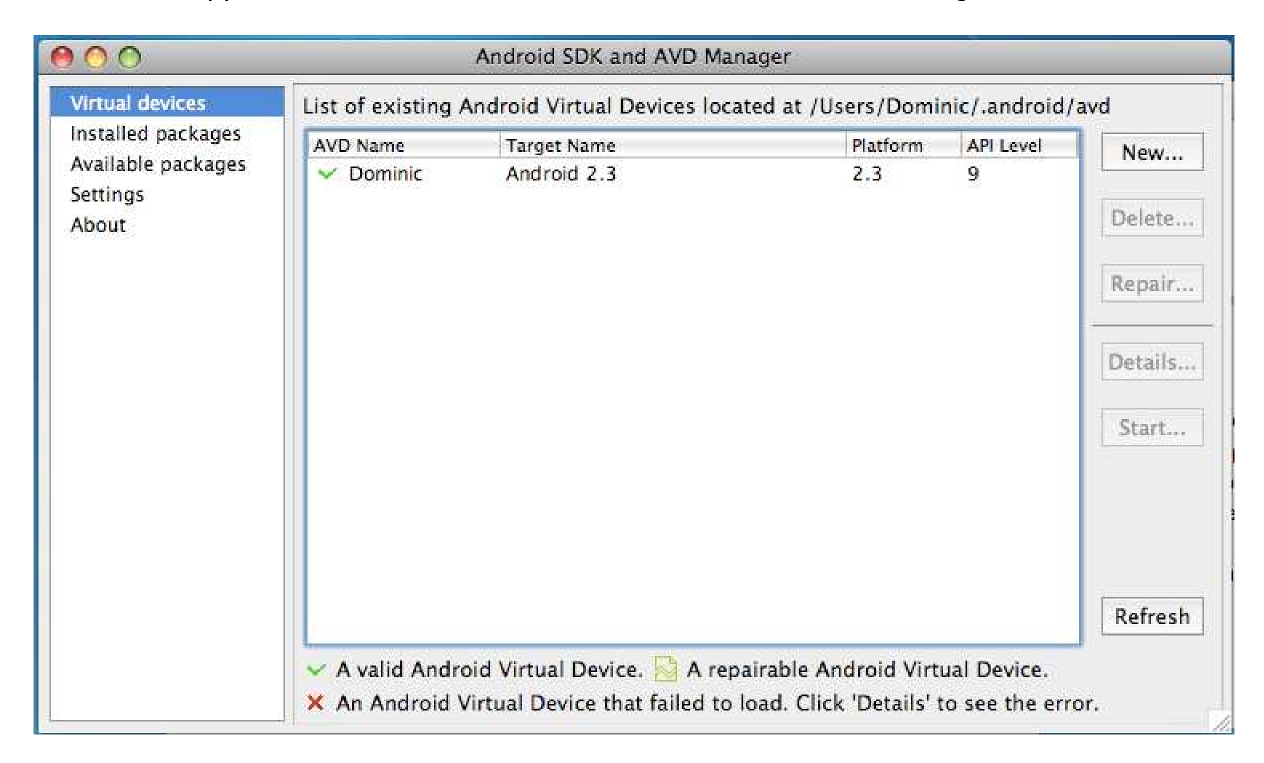
http://developer.android.com/sdk/index.html

Dieses „Starterpaket“ enthält u. a. Tools zum Debuggen und Testen Ihrer Anwendung. Das wichtigste Tool ist der Emulator, der Ihnen ermöglicht ein komplettes Android-Betriebssystem auf Ihrem Rechner nachzuahmen.
Wollen Sie ihre Anwendung direkt auf einem Android-Gerät debuggen, brauchen Sie unter Windows spezielle Treiber, die in der Windows Version enthalten sind. Wenn Sie auf Mac OS X oder Linux entwickeln, brauchen Sie keine speziellen Treiber.
Im Paket gibt es als Einführung in Ihre ersten Anwendungen Beispielcodes und fertige Anwendungen für jede Android-Entwicklungsplattform.

Die SDK enthält eine lokale Kopie der neusten Dokumentation für das Android-API Framework (welche online unter http://developer.android.com/index.html zu erreichen ist).

19.2.2. Emulator erstellen

Um ein Emulator zu erstellen, auf den Sie später debuggen, starten Sie den „Android SDK and AVD Manager". Diesen Manager brauchen Sie, um Emulator oder neue Komponente zu installieren oder vorhandene Komponenten zu upgraden.
Unter *<Ihr SDK-Verzeichnis>/tools/* finden Sie das Tool *„android"* das Sie per Konsole oder mit einem Doppel-Klick ausführen können; nun öffnet sich dieser Manager.



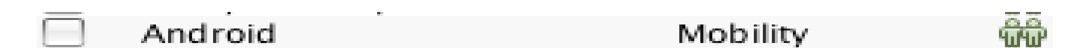
Virtual devices / *New...* - erstellen Sie ein neuen Emulator.
/ *Start...* – starten einen Emulator

19.2.3. Android Pluging in Netbeans installieren

Um Android in Netbeans einzurichten, fügen Sie in der Netbeans-IDE unter *Tools/Plugins/Settings* folgendes Update Center hinzu.

Name: nbandroid
URL: http://kenai.com/downloads/nbandroid/updates.xml

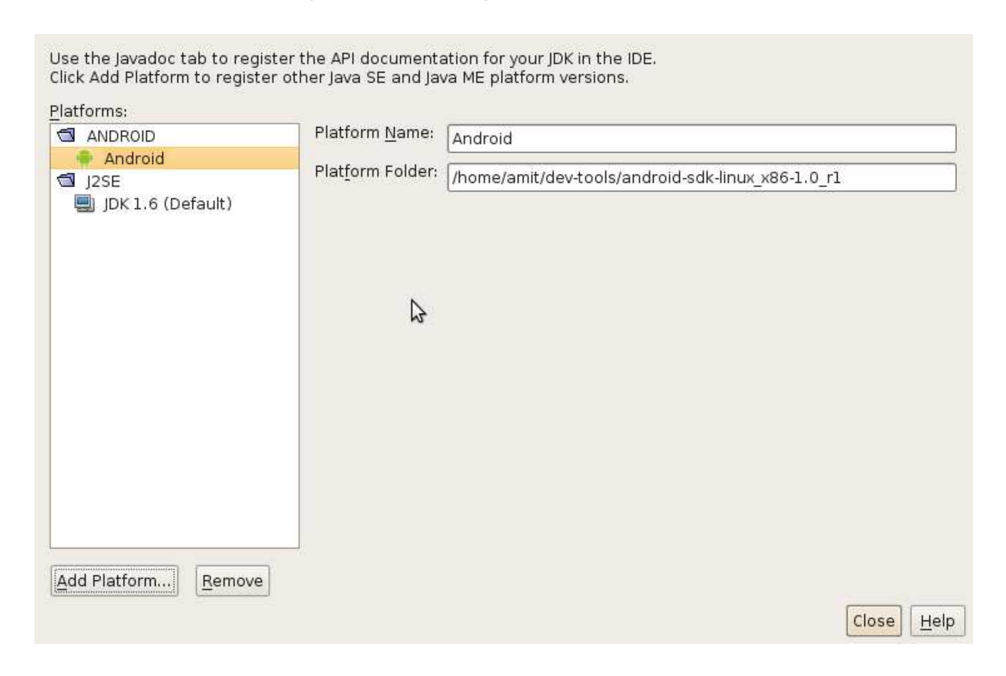
Nun steht unter *Tools/Plugins/Avaible Plugins* das Android-Plugin zum Installieren bereit.



Nachdem Sie das Android-Plugin installiert haben müssen die Android-Platformen in Netbeans hinzugefügt werden. Dies funktioniert unter *Tools/Java Platforms*



Wählen Sie hier *Google Android Open Handheld Platform*.



Die Android-Platformen befinden sich in Ihrem SDK-Verzeichnis.

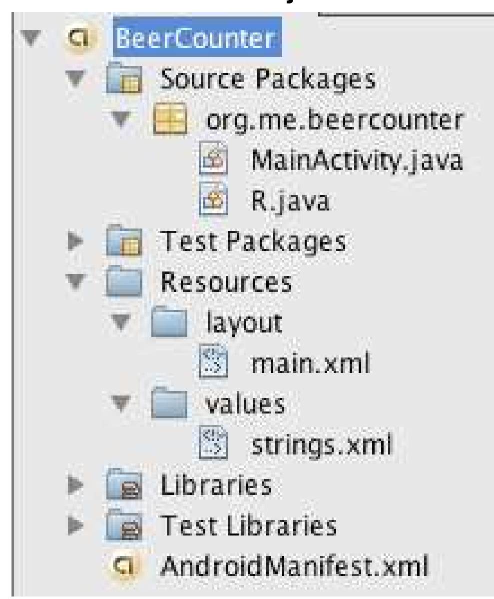
Unter *New Project.../Android* können Sie nun eine neue Android Application auswählen.

19.3. HelloWorld Anwendung mit Android (Beer Counter)

Im folgenden Abschnitt wird gezeigt, wie man eine Android Anwendung entwickelt. Hierzu wird die Anwendung „Beer Counter“ programmiert, dieses App soll beim Berühren eines Buttons zählen, wie viel Bier man getrunken hat.

Erstellen Sie unter *New Project.../Android/ -> „Androi*d Application“ ein neues Android-Projekt.

19.3.1. Android Projektstruktur



MainActivity.java - ist die Klasse der Haupt-Activity

R.Java – Resourcen-ID Definition (diese Datei wird automatisch erstellt und erweitert. Sie darf nicht editiert werden.)

Resources – Ordner für Resource-Dateien wie z. B. Bilder, Layouts, Strings, etc.

Main.xml – Layout-Definition für View der Haupt-Activity

Strings.xml – String-Definition

AndroidManifest.xml – definiert Infos wie Name, Logo und Haupt-Activity

19.3.2. Layout festlegen

In der **Resources/layout/main.xml** wird das Layout der Haupt-Activity definiert. In *<AbsoulteLayout ...* > wird mit *android:background="@drawable/bier"* der Hintergrund festgelegt, dazu wurde vorher im Ordner Resources ein neuer Ordner namens *„drawable"* angelegt und das Bild *„bier.jpg"* dort gespeichert. Mit *<Button ..></Button>* definieren wir einen Button. Damit wir später auf dem Button zu greifen können, wird mit *@+id/count* eine ID in der *R.java* hinterlegt.



In Netbeans 6.9 ist leider noch keine GUI zum Erstellen der Oberfläche integriert. Wenn Sie aber lieber Ihre Oberflächen mit einer GUI erstellen wollen, bietet sich hierfür das Programm DroidDraw an. *(unter* www.droiddraw.org/ *).*

19.3.3. R.java

Wird automatisch erstellt; hier wird für jede Resource ein Konstanter-Integer-Wert hinterlegt. Diese Datei wird nie editiert, man braucht sie eigentlich nicht mal einzusehen.

```
public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int bier=0x7f020000;
        public static final int icon=0x7f020001;
    }
    public static final class id {
        public static final int count=0x7f050001;
        public static final int widget0=0x7f050000;
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
    public static final class string {
        public static final int app_name=0x7f040000;
    }
}
```

Dadurch, dass ein neuer Ordner ***„drawable“*** erstellt wurde, wird automatisch in der R.java Datei eine neue Klasse drawable mit allen Ressourcen die in diesem Ordner liegen., angelegt. Des Weiteren wurde der Button in der Klasse ***id*** hinterlegt.

19.3.4. MainActivity.java

Der Haupt-Activity wird in der onCreate()-Methode mit *setContentView(R.layout.main)* das oben erstellte Layout übergeben*. R.layout.main* ist die Konstante aus der R.java Datei, sie referenziert das in main.xml definierte Layout.

```
public class MainActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        setContentView(R.layout.main);
    }
}
```

Nun wird in der MainActivity der Button Count mit ***findViewById(R.id.count)*** initialisiert. Danach wird ein OnClickListener auf den Button gesetzt. Dieser merkt, wenn man über den Touchscreen auf den Button drückt. Netbeans liefert sofort alle Methoden, die implementiert werden müssen: In diesem Fall die *onClick(View v)* Methode. In dieser Methode wird nun die *Integer Variable* count inkrementiert und der Text vom Button *Count* wird neu gesetzt.

```
final Button Count = (Button)findViewById(R.id.count);
        Count.setOnClickListener(new View.OnClickListener() {
         public void onClick(View v) {
               count++;
               Count.setText("Bier "+ count);
            }
        });
```

19.3.5. AndroidManifest.xml

In der AndroidManifest.xml wird die Main-Activity definiert, also die Klasse, die als erstes beim Starten der Anwendung aufgerufen wird. Durch android:label="Beer Counter" wird dem Label von der Activity ein anderer Name gegeben. Dann wird durch android:icon="@drawable/icon" der Anwendung im Menu ein neues Logo zugeteilt, sonst würde das Standard Android Logo gesetzt werden.

```
<?xml version="1.0" encoding="UTF-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
     package="org.me.beercounter">
    <application>
         <activity android:name=".MainActivity" android:label="Beer
Counter" android:icon="@drawable/icon">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category
android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>
</manifest>
```

19.3.6. Debuggen

Beim Debuggen der Anwendung, wird nun das AVD (Android Virtual Devices) automatisch gestartet. Wenn Sie direkt auf Ihrem Androidphone debuggen, müssen Sie auf Ihrem Gerät erst das USB-Debugging aktivieren. Diesen können Sie auf Ihrem Androidphone unter *Einstellung/Anwendungen/Entwicklung/USB-Debugging* einschalten. Wenn Sie nun Ihr Projekt debuggen und Ihr Smartphone über USB angeschlossen ist, wird Ihre Anwendung direkt auf Ihrem Gerät installiert und gestartet.

Nun können Sie im laufe des Abends zählen, wie viel Bier Sie getrunken haben.







19.4. Quellen

http://developer.android.com/
http://www.androidpit.de/
http://www.androidbuch.de/
http://wiki.netbeans.org/IntroAndroidDevNetBeans
http://de.wikipedia.org/wiki/Android_%28Betriebssystem%29

19.5. Quellcode

MainActivity.java

```java
package org.me.androidapplication1;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class MainActivity extends Activity {

    private int count = 0 ;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        setContentView(R.layout.main);
        final Button Count = (Button)findViewById(R.id.count);
        Count.setOnClickListener(new View.OnClickListener() {
         public void onClick(View v) {
               count++;
               Count.setText("Bier "+ count);
            }
        });
    }
}
```

main.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout
    android:id="@+id/widget0"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="@drawable/bier"
    xmlns:android="http://schemas.android.com/apk/res/android"
    >
    <Button
        android:id="@+id/count"
        android:layout_width="171px"
        android:layout_height="154px"
        android:text="Count "
        android:layout_x="65px"
        android:layout_y="118px"
        >
    </Button>
</AbsoluteLayout>
```

AndroidManifest.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
     package="org.me.androidapplication1">
    <application>
         <activity android:name=".MainActivity" android:label="Beer Counter"
android:icon="@drawable/icon">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>
</manifest>
```

20. JavaFX (mobile) – Tobias Schulz, 10.01.2010

20.1. Was ist JavaFX?

Java FX wurde als Open Source Framework 2008 von Sun veröffentlicht. Es dient zur Entwicklung von Web-, Desktop- und mobilen Anwendungen. Dieses Rich Internet Application Framework steht in Konkurrenz zu Adobe Flash und Microsoft Silverlight. JavaFX braucht nur die Java SE Runtime, die auf allen Rechnern, auf denen Java läuft, schon vorhanden ist. Damit erreicht JavaFX 75% aller PCs. Adobe Flash erreicht dagegen 100% was für die meisten an Anbieter eher attraktiv ist. Microsoft Silverlight erreicht immerhin noch 50%. Angesichts dieser Zahlen ist es nicht leicht für JavaFX sich zu etablieren. Vor allem da viele Anbieter neben Flash in Zukunft auf HTML 5 setzen. JavaFX legt ein besonderes Augenmerk auf Multimedia und 3D so wie 2D Grafik. Viele Eigenschaften sind so angelegt, das es für Designer besonders einfach ist mit JavaFX zu arbeiten. Es war vorher auch selten so einfach Apps fürs Web zu entwickeln. Man braucht sehr wenig Vorkenntnisse und der Benutzer kann die eigene App jederzeit „Downloaden". Sie muss nur aus dem Browser raus gezogen werden.
JavaFX ist ein sehr einfaches RIA Framework zum entwickeln von Apps für die verschiedensten Plattformen (TV, PC, WEB, ...). Es ist besonders für Multimedia Apps geeignet und unterstützt 2D und 3D Grafik.

20.2. Warum JavaFX?

JavaFX ermöglicht es sehr einfach die Oberfläche eines Apps Interaktiv zu gestalten. mit Java ist es sehr umständlich eine Oberfläche zu erstellen die sich nachträglich verändern soll. Oder Objekte(z.B. Bilder) die sich transformieren. Es werden ausserdem sehr viele verschiedenen Grafik und Video Formate unterstützt. Es gibt sogar ein Plugin für Adobe Photoshop, welches einfach die Grafiken die man erstellt direkt für JavaFX anpasst. Man kann mit Java FX daher sehr einfach einen Media Player in seine Homepage einbinden ohne viel dafür können zu müssen oder einfach sehr schöne Menüs mit sich bewegenden Objekten erstellen. Diese Apps können auch mit dem Webserver kommunizieren. Ein Einfaches Spiel zu entwickeln ist mit JavaFX ebenso kein Problem.

20.3. Warum nicht JavaFX?

JavaFX hat mit mehreren Problemen zu kämpfen. Das Framework läuft in einer Java Virtual Machine sehr langsam. Das merkt der Benutzer besonders stark bei Apps auf einer Homepage wenn neben einer längeren Ladezeit der Inhalte die JVM auch noch mal deutliche Ladezeiten hat. Ausserdem muss der Benutzer das Ausführen von Apps bestätigen. Viele Benutzer die auf Sicherheit beim PC bedacht sind drücken daher auf nein. Man kann sich nie sicher sein was die Java App wirklich auf dem eigenen Rechner macht da man ihr nur „alle Rechte oder gar keine Rechte" geben kann. Keine Rechte bedeutet gleichzeitig auch kein Ausführen der JavaFX App. Flash ist da aktuell deutlich im Vorteil und viel Komfortabler für den Benutzer. Aber nicht nur Flash macht es besser, immer mehr Anbieter setzen in naher Zukunft auf HTML5, welches Flash den Rang ablaufen könnte. Damit hat JavaFX im Bereich Multimedia 2 „übermächtige" Konkurrenten. Das ist aber nicht das einzige Problem von JavaFX. JavaFX wurde besonders viel Wert drauf gelegt, das es auf möglichst vielen Endgeräten läuft(PC, TV, Handy, ...). Man hatte sich besonders viel von Handy Apps versprochen, wo JavaFX aber dran vorbei geschossen ist. LG, Sony und ein paar andere Firmen wollten mit Sun zusammen JavaFX auf Android bringen. Dieser Plan schlug fehl. In der Praxis sind JavaFX mobile Anwendungen viel zu langsam auf diesen Endgeräten und nicht Praxisgerecht. Dort wurde Java FX zum Opfer seiner eigenen Vielfältigkeit. Google(Android) und Apple(IPhone) haben beide einen anderen Weg verfolgt, der sich aktuell als der bessere herausgestellt hat. Google und Apple haben für ihre Mobile Devices Programmier sprachen angepasst. Bei dieser Anpassung wurde sehr viel wert auf das eigentliche Gerät selbst gelegt und nicht auf eine möglichst große Kompatibilität. Das

Ergebnis ist, dass Apps auf dem IPhone und Android so schnell laufen wie auf keinem Smartphone davor. Programme haben fast keine Ladezeiten, man klickt in einer WebApp und es kommt sofort eine Reaktion, usw.. Dieses Ergebnis lässt sich mit JavaFX nicht erzielen. Es läuft im Gegensatz dazu eher sehr langsam und Aktionen werden meist verspätet verarbeitet. Dazu kommt auch noch, dass Sun von Oracle aufgekauft wurde und dadurch sich der Entwicklungsschwerpunkt sich deutlich verschoben hat in Richtung Desktop Anwendungen und Web Apps. Oracle steht mit Google ausserdem in einem Patentstreit wegen der Sandbox in Android. Dieses ist sicherlich auch nicht förderlich für eine Unterstützung von JavaFX auf einem Smartphone.

20.4. JavaFX 2011

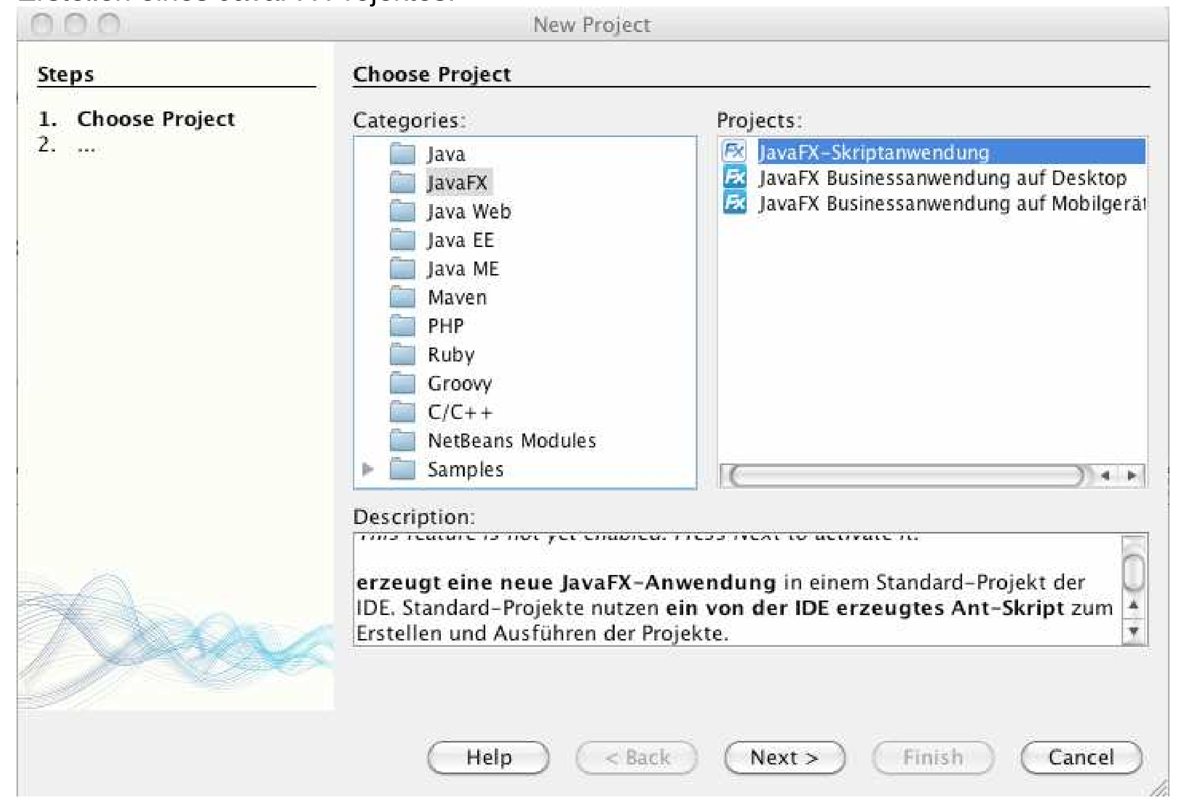
Letztes Jahr verkündete Oracle, dass JavaFX in Version 2 im 3.Quartal 2011 erscheint. JavaFX2.0 soll auch wieder OpenSource werde laut Oracle. JavaFX2.0 enthält einige sehr große Änderungen. Es fällt z.B. JavaFX Script weg. Mit JavaFX Script wurden bisher JavaFX Apps erstellt. Anstelle dessen wird dann die Java-API genutzt. Dadurch sollen sich andere Sprachen auch nutzen lassen und auch Performance gewonnen werden. Des weiteren erhält JavaFX2.0 eine hardwarebeschleunigte Rendering-Engine und neue User-Interface-Steuerelemente. Es wird auch das Zusammenspiel mit HTML5 massiv verbessert. Oracle gibt auch an das JavaFX2.0 insgesamt schlanker und vor allem deutlich schneller werden soll. In der API sollen dann Java-2D/OpenGL-/Hotspot-VM-Stack als auch JavaScript und HTML5 enthalten sein.

Zu dem Mobile Bereich von JavaFX wurde sich nicht geäussert. Ansonsten kann man zumindest wieder Licht am Ende des Tunnels für JavaFX sehen.

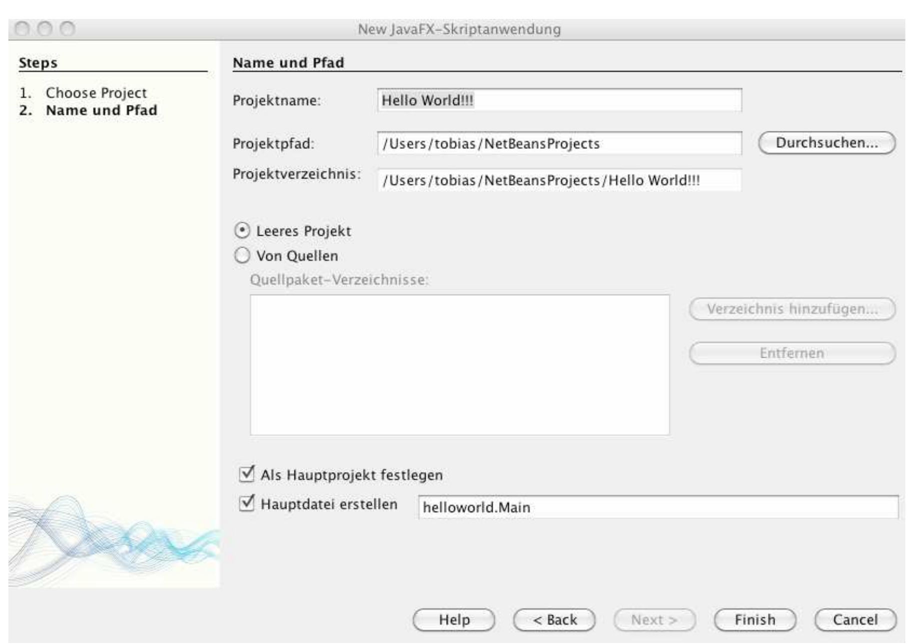
20.5. Beispiele

20.5.1. Hello World

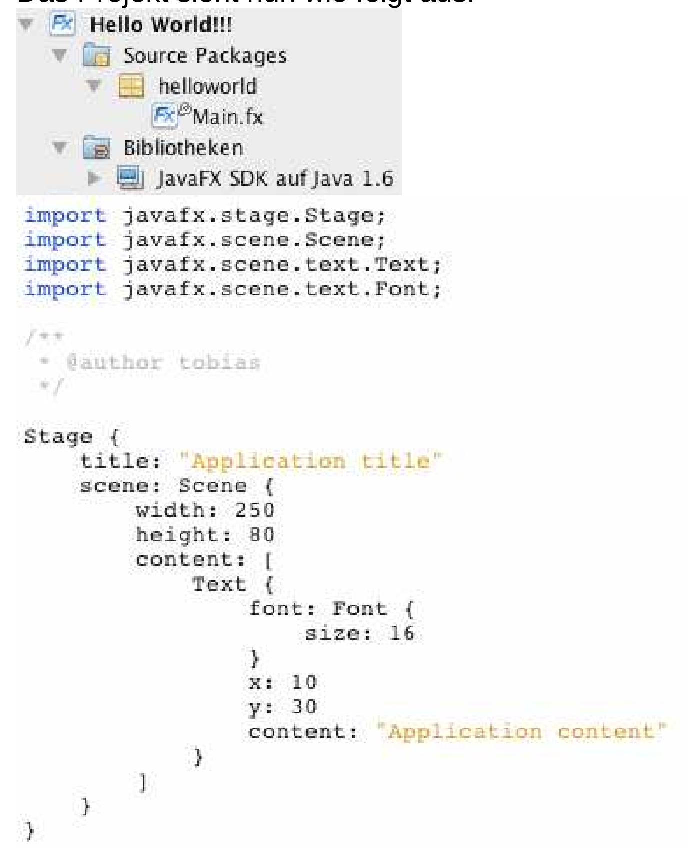
Erstellen eines JavaFX Projektes:



Categorie: “JavaFX” >>> Projects: “JavaFX-Scriptanwendung” >>> “Next”



Projektname: “Hello World!!!” >>> “Finish”
Das Projekt sieht nun wie folgt aus:




```
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.text.Text;
import javafx.scene.text.Font;

/**
 * @author tobias
 */

Stage {
    title: "Application title"
    scene: Scene {
        width: 250
        height: 80
        content: [
            Text {
                font: Font {
                    size: 16
                }
                x: 10
                y: 30
                content: "Application content"
            }
        ]
    }
}
```

Stage ist das Grundgerüst für JavaFX Script.
Scene beschreibt die Arbeitsumgebung wo man Objekte (Text, buttons, usw.) platzieren kann.
Text ist ein Element zur Darstellung von Text.

Paar Zeilen Code:

```
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.text.Text;
import javafx.scene.text.Font;
import javafx.scene.paint.Color;

Stage {
    title: "Personal Hello"
    scene: Scene {
        width: 250
        height: 80
        content: [
            Text {
                fill: Color.BLUE
                font: Font {
                    size: 16
                }
                x: 25
                y: 25
                content: "Hello World!!!"
            }
        ]
    }
}
```

Programm kompilieren und starten

Personal Hello

Hello World!!!

20.5.2. Mediaplayer



Für einen Mediaplayer unter JavaFX brauch man nicht viel Code:

```
def vid = "file:/Users/tobias/NetBeansProjects/video/src/test/video-clip.flv";

var isPlaying = false;

def myMediaPlayer = MediaPlayer {
        media: Media {source: vid}
}

def myMediaView = MediaView {

    onMousePressed: function(event){

        if(isPlaying){
            myMediaPlayer.pause();
        }else{
            myMediaPlayer.play();
        }

            isPlaying = not isPlaying;
    }

    mediaPlayer: myMediaPlayer
}
function run(){
    Stage {
        title: "JavaFX Video Player"
        scene: Scene {
            width: 600
            height: 350

            fill: Color.BLACK
            content: [myMediaView]
        }
    }
}
```

Diese paar Zeilen Code reichen um einen Multimedia Player mit Play/Pause zu erstellen. Diesen kann man nun ohne viel Arbeit anderen auf der eigenen Homepage zur Verfügung stellen.

20.6. Schlusswort

JavaFX ist eine sehr mächtiges Framework für die RIA Programmierung mit vielen Schwächen. Die Geschwindigkeit ist dabei das größte Problem von JavaFX. Dadurch konnte JavaFX nie wirklich Fuß fassen in der Welt der Browser. Obwohl das Framework den Übergang zwischen Web-Apps und Desktop-Apps nahezu perfekt meistert wird es sehr selten genutzt. Dazu kommt auch noch, das Oracle JavaFX1.3 lange schleifen gelassen hat und dieses nun komplett überarbeitet und damit einen Neustart versucht. Dieses hat zu folge das JavaFX vielleicht wieder in dem Markt mitkämpfen kann und es zu einer sehr attraktiven Umgebung zum entwickeln von Web- und Desktop-Apps wird.
Im Moment kann man nur ganz klar sagen, das Adobe's Flash stärker und weiter verbreitet ist und es für JavaFX mit HTML5 nicht einfacher wird.

20.7. Tutorials die auf dieses Thema aufbauen:

http://download.oracle.com/javafx/index.html
http://www.cppjj.com/JavaFX/javafx.htm
http://jfx.wikia.com/wiki/Tutorials

20.8. Quellen:

http://download.oracle.com/javafx/index.html
http://www.heise.de
http://www.wikipedia.org
http://www.javafx.com

21. Veränderungen am Programm Fotosorter- Paul Bernasch

Begleitend zum Seminar wurde ein Beispielprogramm verbessert. Nachdem der Code in 3 Klassen aufgeteilt wurde (gemäß des erstellten UML-Diagramms) wurden noch weitere Verbesserungen mit Netbeans vorgenommen.

21.1.1. Vereinfachung des Datumformates in Timestamp

Wir schrieben eine Funktion Timestamp die uns aus einem Kalenderobjekt oder einem Bild, einen formatierten String ausgibt, so dass wir nach diesem String die Bilder sortieren können. In der ersten Version des Fotosorters wurde in Timestamp jeder Wert einzeln aus dem Datum ausgelesen und in den String **ret** gespeichert.

Der Code sah zu dem Zeitpunkt wie folgt aus

```
private String timeStamp(File file, int years, int months, int days, int hours, int
mins, boolean save) {
        String ret = "";
        try {
            if (file.exists()) {
                GregorianCalendar cal = new GregorianCalendar();
                cal.setTime(new Date(file.lastModified()));
                cal.add(Calendar.YEAR, years);
                cal.add(Calendar.MONTH, months);
                cal.add(Calendar.DAY_OF_MONTH, days);
                cal.add(Calendar.HOUR_OF_DAY, hours);
                cal.add(Calendar.MINUTE, mins);
                int mo = (cal.get(Calendar.MONTH) + 1);
                int da = cal.get(Calendar.DATE);
                int ho = cal.get(Calendar.HOUR_OF_DAY);
                int mi = cal.get(Calendar.MINUTE);
                ret += cal.get(Calendar.YEAR) + ".";
                ret += (mo < 10) ? "0" + mo : "" + mo;
                ret += (da < 10) ? ".0" + da : "." + da;
                ret += (ho < 10) ? " 0" + ho : " " + ho;
                ret += (mi < 10) ? ":0" + mi : ":" + mi;
                if (save) {
                    file.setLastModified(cal.getTimeInMillis());
                }
            } else {
                System.out.println("File does not exists!");
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return ret;
    }
```

Durch die Veränderung in Timestamp ist der Code auf zehn Zeilen reduziert und zwar mit Hilfe von string.format.
String.format holt sich die Werte aus einem Kalenderobjekt, hier **cal** und setzt diese in den String **ret** ein. Dieser String **ret** wird dann übergeben.

Die Funktion TimeStamp sieht damit wie folgt aus:

```
  public static String timeStamp(GregorianCalendar cal) {

        String ret = String.format("%1$tY.%1$tm.%1$td %1$tH:%1$tM", cal);

        Logger.getLogger(FotoFiles.class.getName()).log(Level.INFO, ret);

        return ret;

    }
```

21.1.2. Einführung von Fileseparator und Anpassung der URL von Hilfe

Wir haben die Separator-Methode verwendet, mit Hilfe dieser Methode können wir beim Umformen in einen String, aus oder in einen abstrakten Pfadnamen einheitlich abspeichern. Die einzelnen Elemente werden durch einen Separator Character getrennt, und dieser ist stark systemabhängig. Durch den Fileseparator haben wir ein einheitliches Trennungszeichen welches in der fileseperator.properties festgelegt ist.
Anhand des Fileseparators haben wir die URL für den Hilfeaufruf normieren können.
In der ersten Version des Fotosorters hatten wir als Trennungszeichen \\ dies ist sehr systemabhänging und es könnte passieren das einige Systeme den Pfad nicht verarbeiten können hier kommt in der neuen Version des Fotosorters der Fileseparator ins Spiel wir ersetzen im Code das Trennungszeichen \\ durch den FileSeparator in der replace-Funktion und erhalten somit eine für alle Systeme einheitliche URL für die Hilfe.htm.

Die beiden Codes im Vergleich :

```
userDir=userDir.replace("\\", "/");     // replace Funktion ohne Fileseparator
Desktop.getDesktop().browse( new URI(userDir+"/help.html") );

userDir = userDir.replace(fileSeparator, "/");//replace Funktion mit Fileseparator
userDir=userDir.replace(" ", "%20");
Desktop.getDesktop().browse(new URI("file:///"+userDir + "/help.html"));
```

21.1.3. Vereinfachungen der FocusLost-Methoden

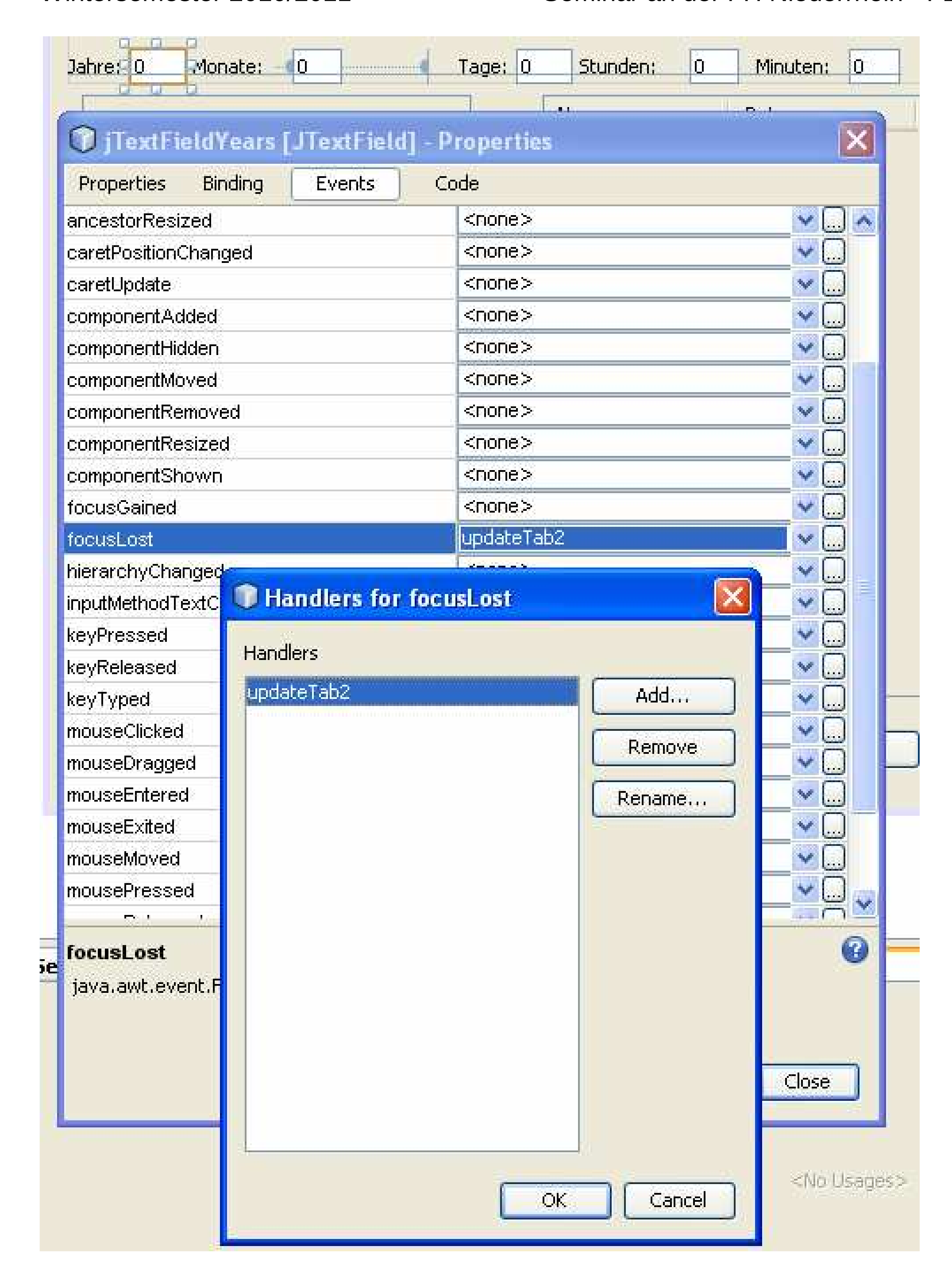
In der Ursprungsversion des Fotosorters wurde für jedes FocusLost- Event eine eigene Methode mittels des Netbeans-Wizard erzeugt.

```
523    private void jTextFieldYearsFocusLost(java.awt.event.FocusEvent evt) {
524      buildTab2(false);
525    }
526
527    private void jTextFieldMonthsFocusLost(java.awt.event.FocusEvent evt) {
528      buildTab2(false);
529    }
530
531    private void jTextFieldMinsFocusLost(java.awt.event.FocusEvent evt) {
532      buildTab2(false);
533    }
```

Da diese Methoden alle den gleichen Code enthalten wurde im Kurs gezeigt, wie dies vereinfacht werden kann.
Hierzu wurde eine neue Methode erstellt:

```
437    private void updateTab2(java.awt.event.FocusEvent evt) {
438      buildTab2(false);
439    }
```

Nun müssen nur noch die Textfelder mit dieser Methode verbunden werden. Hierzu klicken wir im Designmodus mit der rechten Maustaste in das entsprechende Textfeld und wählen Properities. Dort wählen wir Events. Und wählen für den Handler die neue Methode aus (die ursprünglichen Methoden löschen) und fertig. Somit wird der Code deutlich kürzer und Redundanzen entfallen.

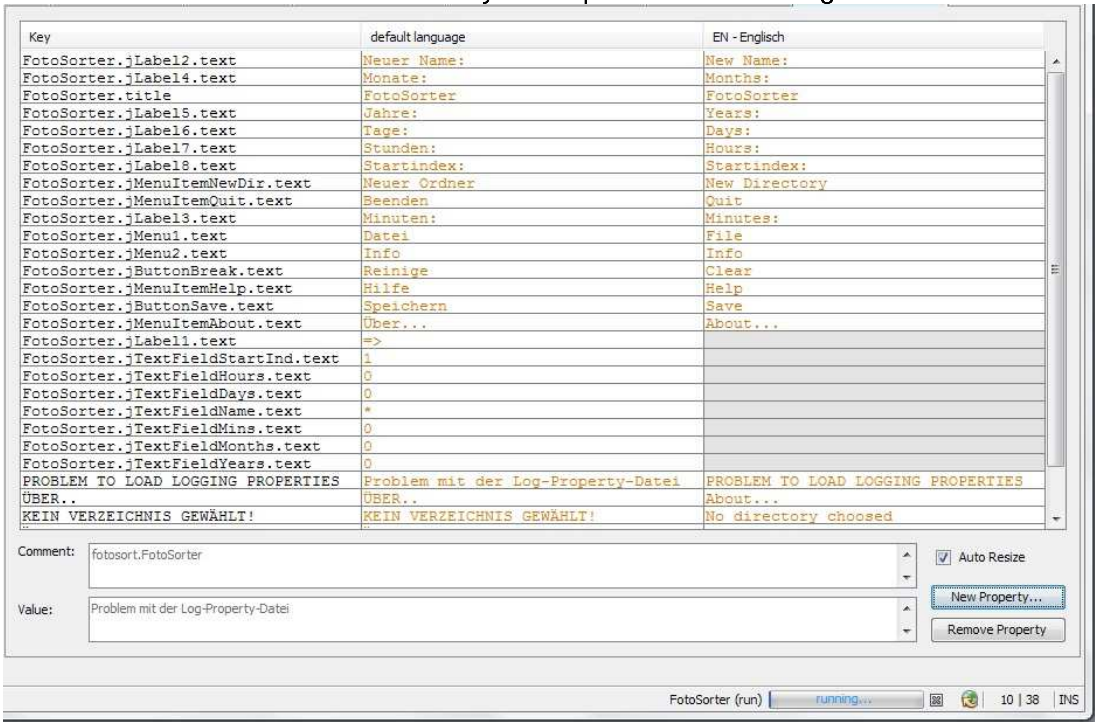


21.1.4. Internationalisierung des Fotosorters

Wir haben Fotosorter internationalisiert, in dem wir eine Bundle.properties-Datei erstellten (wie dies funktioniert ist im Kapitel über Internationalisierung nach zu lesen).

In der Bundle.properties-Datei wurde dann den einzelnen Keys, die durch die Elemente der Benutzeroberfläche definiert sind, als erstes die Values/ Werte der default-Sprache zugewiesen, welche in unserem Deutsch ist .
Als nächster Schritt wurde die Bundle_en_En.properties-Datei erstellt und bearbeitet hier wurde den Keys die Values der englischen Sprache hinzugefügt.

Das Fenster für das zuweisen der key/value paare sieht wie folgt aus :



Key	default language	EN - Englisch
FotoSorter.jLabel2.text	Neuer Name:	New Name:
FotoSorter.jLabel4.text	Monate:	Months:
FotoSorter.title	FotoSorter	FotoSorter
FotoSorter.jLabel5.text	Jahre:	Years:
FotoSorter.jLabel6.text	Tage:	Days:
FotoSorter.jLabel7.text	Stunden:	Hours:
FotoSorter.jLabel8.text	Startindex:	Startindex:
FotoSorter.jMenuItemNewDir.text	Neuer Ordner	New Directory
FotoSorter.jMenuItemQuit.text	Beenden	Quit
FotoSorter.jLabel3.text	Minuten:	Minutes:
FotoSorter.jMenu1.text	Datei	File
FotoSorter.jMenu2.text	Info	Info
FotoSorter.jButtonBreak.text	Reinige	Clear
FotoSorter.jMenuItemHelp.text	Hilfe	Help
FotoSorter.jButtonSave.text	Speichern	Save
FotoSorter.jMenuItemAbout.text	Über...	About...
FotoSorter.jLabel1.text	=>	
FotoSorter.jTextFieldStartInd.text	1	
FotoSorter.jTextFieldHours.text	0	
FotoSorter.jTextFieldDays.text	0	
FotoSorter.jTextFieldName.text	*	
FotoSorter.jTextFieldMins.text	0	
FotoSorter.jTextFieldMonths.text	0	
FotoSorter.jTextFieldYears.text	0	
PROBLEM TO LOAD LOGGING PROPERTIES	Problem mit der Log-Property-Datei	PROBLEM TO LOAD LOGGING PROPERTIES
ÜBER..	ÜBER..	About...
KEIN VERZEICHNIS GEWÄHLT!	KEIN VERZEICHNIS GEWÄHLT!	No directory choosed